R3465 SERIES OPT15 SPECTRUM ANALYZER

PROGRAMMING MANUAL

OPERATIONS

ADVANTEST®
ADVANTEST CORPORATION

ADVANTEST®
ADVANTEST CORPORATION

R3465 Series OPT15

Spectrum Analyzer

PROGRAMMING MANUAL

MANUAL NUMBER OEA01 9701

This manual is for the following models.
-R3465
-R3272
-R3263

Before reselling to other corporations or re-exporting to other countries, you are required to obtain permission from the Japanese Government under its Export Control Act.

© 1995 ADVANTEST CORPORATION
All rights reserved.

First printing November 14, 1995
Printed in Japan

ADVANTEST

ADVANTEST CORPORATION

R3465 Series OPT15

Spectrum Analyzer

PROGRAMMING MANUAL

MANUAL NUMBER OEAP1 E01

This manual is for the following models.

-R3465
-R3272
-R3263

Before reselling to other corporations or re-exporting to other countries, you are required to obtain permission from the Japanese Government under its Export Control Act.

© 1995 ADVANTEST CORPORATION
All rights reserved.

First printing November 1st, 1995
Printed in Japan

Safety Summary

To ensure thorough understanding of all functions and to ensure efficient use of this instrument, please read the manual carefully before using. Note that Advantest bears absolutely no responsibility for the result of operations caused due to incorrect or inappropriate use of this instrument.

If the equipment is used in a manner not specified by Advantest, the protection provided by the equipment may be impaired.

■Warning Labels

Warning labels are applied to Advantest products in locations where specific dangers exist. Pay careful attention to these labels during handling. Do not remove or tear these labels. If you have any questions regarding warning labels, please ask your nearest Advantest dealer. Our address and phone number are listed at the end of this manual.

Symbols of those warning labels are shown below together with their meaning.

DANGER: Indicates an imminently hazardous situation which will result in death or serious personal injury.

WARNING: Indicates a potentially hazardous situation which will result in death or serious personal injury.

CAUTION: Indicates a potentially hazardous situation which will result in personal injury or a damage to property including the product.

■Basic Precautions

Please observe the following precautions to prevent fire, burn, electric shock, and personal injury.

- Use a power cable rated for the voltage in question. Be sure however to use a power cable conforming to safety standards of your nation when using a product overseas. Do not place anything heavy on top of the power cable.

- When inserting the plug into the electrical outlet, first turn the power switch OFF and then insert the plug as far as it will go.

- When removing the plug from the electrical outlet, first turn the power switch OFF and then pull it out by gripping the plug. Do not pull on the power cable itself. Make sure your hands are dry at this time.

- Before turning on the power, be sure to check that the supply voltage matches the voltage requirements of the instrument.

- Be sure to plug the power cable into an electrical outlet which has a safety ground terminal. Grounding will be defeated if you use an extension cord which does not include a safety ground terminal.

- Be sure to use fuses rated for the voltage in question.

- Do not use this instrument with the case open.

- Do not place objects on top of this product. Also, do not place flower pots or other containers containing liquid such as chemicals near this product.

- When the product has ventilation outlets, do not stick or drop metal or easily flammable objects into the ventilation outlets.

- When using the product on a cart, fix it with belts to avoid its drop.

- When connecting the product to peripheral equipment, turn the power off.

■Caution Symbols Used Within this Manual

Symbols indicating items requiring caution which are used in this manual are shown below together with their meaning.

DANGER : Indicates an item where there is a danger of serious personal injury (death or serious injury).

WARNING : Indicates an item relating to personal safety or health.

CAUTION : Indicates an item relating to possible damage to the product or instrument or relating to a restriction on operation.

■Safety Marks on the Product

The following safety marks can be found on Advantest products.

: ATTENTION - Refer to manual.

: Protective ground (earth) terminal.

: DANGER - High voltage.

: CAUTION - Risk of electric shock.

■Precautions when Disposing of this Instrument

When disposing of harmful substances and batteries, be sure dispose of them properly with abiding by the state-provided law.

Harmful substances: (1) PCB (polycarbon biphenyl)
(2) Mercury
(3) Ni-Cd (nickel cadmium)
(4) Other
Items possessing cyan, organic phosphorous and hexadic chromium and items which may leak cadmium or arsenic (excluding lead in solder).

TABLE OF CONTENTS

1. INTRODUCTION 1-1

2. OPERATING BASICS 2-1

2.1 Overview of Operation 2-1
2.2 Panel Operation 2-1
2.2.1 Inputting, Executing and Ending Program 2-1
2.2.2 Data Input Keys 2-2
2.2.3 Function Keys 2-2
2.3 Memory Card 2-3
2.3.1 Usable Memory Card 2-3
2.3.2 Memory Card Specifications 2-4
2.3.3 Note on Handling the Memory Card 2-4
2.3.4 Insertion and Ejection of Memory Card 2-5
2.4 File Management 2-6
2.4.1 Summary 2-6
2.4.2 File Management 2-7
2.4.3 Storing Files 2-8
2.4.4 Loading Files 2-8
2.4.5 Erasing Files 2-9
2.4.6 Changing File Name 2-9
2.5 Screen Layout 2-10
2.6 External Keyboard 2-12
2.6.1 Connecting External Keyboard 2-13

3. BASIC COMMANDS 3-1

3.1 Various Commands 3-1
3.1.1 List of Command Function 3-2
3.1.2 List of Command Syntax 3-3
3.1.3 Precautions Common to All Commands 3-4
3.2 Command Grammar and Application 3-5
1. Program Input 3-5
2. CAT 3-5
3. CHDIR 3-6
4. CHKDSK 3-7
5. CONT 3-8
6. CONTROL 3-9
7. COPY 3-11
8. DEL 3-12
9. END 3-13
10. GLIST 3-14

11. GLISTN 3-15
12. INDENT 3-16
13. LIST 3-17
14. LISTN 3-18
15. LLIST 3-19
16. LLISTN 3-20
17. LOAD 3-21
18. MERGE 3-22
19. PAUSE 3-24
20. PRINTER 3-25
21. PURGE 3-25
22. PWD 3-26
23. REN 3-27
24. RENAME 3-28
25. RUN 3-29
26. SAVE 3-30
27. SCRATCH 3-31
28. STEP 3-32
29. STOP 3-33

4. BASIC STATEMENT 4-1

4.1 Programming Rules 4-1
4.1.1 Program Structure 4-1
4.1.2 Object 4-3
4.1.3 Operators 4-8
4.2 Various Statements 4-12
4.2.1 Statement Function List 4-12
4.2.2 Statement Syntax List 4-14
4.3 Statement Syntax and Use 4-21
1. ABS 4-21
2. ATN 4-22
3. CHR$ 4-23
4. CLEAR 4-24
5. CLOSE 4-25
6. CLS 4-26
7. COLOR 4-27
8. COMMON 4-28
9. CONSOLE 4-30
10. COS 4-31
11. CSRLIN 4-32
12. CSRPOS 4-33
13. CURSOR 4-34

14. DATA 4-35
15. DATE$ 4-36
16. DELIMITER 4-37
17. DIM 4-38
18. DISABLE INTR 4-40
19. DSTAT 4-41
20. ENABLE INTR 4-43
21. ENTER 4-44
22. ENTER USING 4-47
23. ERRM$ 4-50
24. ERRN 4-51
25. EXP 4-52
26. FOR - TO - STEP, NEXT, BREAK, CONTINUE 4-53
27. FRE 4-55
28. GOSUB, RETURN 4-56
29. GOTO 4-58
30. GPRINT 4-59
31. IF-THEN, ELSE, END IF 4-61
32. INKEY$ 4-64
33. INPUT (INP) 4-65
34. INTEGER 4-67
35. INTERFACE CLEAR 4-69
36. LEN 4-70
37. LOCAL 4-71
38. LOCAL LOCKOUT 4-72
39. LOG 4-73
40. LPRINT 4-74
41. NUM 4-75
42. OFF END 4-76
43. OFF ERROR 4-77
44. OFF KEY 4-78
45. OFF SRQ, OFF ISRQ 4-79
46. ON DELAY 4-80
47. ON END 4-81
48. ON ERROR 4-82
49. ON KEY 4-83
50. ON SRQ, ON ISRQ 4-85
51. OPEN 4-86
52. OUTPUT 4-88
53. OUTPUT USING 4-91
54. POS 4-93
55. PRINT [USING] 4-94

96 PRINTER 4-97
97 PRINTF 4-98
98 READ 4-100
99 REAL 4-101
9C REM 4-102
A1 REMOTE 4-103
A2 REQUEST 4-104
A3 RESTORE 4-105
A4 SELECT, CASE, ENS SELECT 4-106
A5 SEND 4-107
A6 SIN 4-109
A7 SPOLL 4-110
A8 SPRINTF 4-111
A9 TIMER 4-112
70 TIME$ 4-113
71 TRIGGER 4-114
72 WAIT 4-115
73 WAIT EVENT 4-116

5. ERROR MESSAGES 5-1

5.1 How to Check Error Message Line Number 5-1
5.2 How to Check Program Current Position 5-1
5.3 Error Message List 5-1

ALPHABETICAL INDEX I-1

LIST OF ILLUSTRATIONS

No.	Title	Page
2-1	Drive Slot for Memory Card	2-5

LIST OF TABLES

No.	Title	Page
2-1	Memory Card Specifications	2-4
4-1	Key Word List	4-2
4-2	Correspondence Table between Full Name and Short Name	4-2
4-3	Alphanumeric Characters	4-5

1. INTRODUCTION

The BASIC language built into the spectrum analyzer is equipped with general-purpose BASIC commands, GPIB control purpose commands, and exclusive built-in functions, enabling the spectrum analyzer to be used for simple configuration of small GPIB systems.

- Command and statement syntax

The syntax for the commands and statements used for this analyzer is explained in Chapters 3 and 4 of this manual with both schematic and descriptive representations for intuitive understanding.

CAUTION

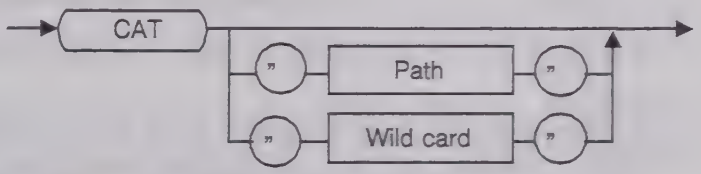
How to read the syntax for commands and statements

(1) Schematic representation

To represent a syntax, the analyzer disassembles it into its elements and connects them with straight lines.

Statements should always be read in the direction of the arrows. If a statement jumps to multiple branches on the way, the analyzer will go to one of them. If a loop is formed in the representation, the loop can be passed any number of times.

Description example:



(2) Meanings of symbols used for descriptive representation

- Part enclosed with symbols [ ]: Indicates that the enclosed item is an option (omissible).
- Part enclosed with symbols < >: Indicates that the enclosed item is not an option (un-omissible).
- Part enclosed with symbols { }: Indicates that the enclosed item is repeatable 0 times or more.
- Symbol | : Indicates "or". (ex. A|B - A or B is selectable.)

Example of representation: CAT ["[Drive name:] [Directory/] [Wild card] "]

(3) Meanings of words used for schematic and descriptive representations

- Numerical value representation expression:
  Any one of numeric value constant, numeric value variable,numeric array variable and expression
- Character string representation expression:
  Expression consisting of character string constant, character-string variable, character-string array variable, and sub-string
- Equipment address: Address of device connected to GPIB

- GPIB mode

The analyzer operates in either of two modes: ADDRESSABLE or CONTROL. The switching between the modes is performed using the CONTROL command or from the front panel.

For the use of the CONTROL command, refer to "3. BASIC COMMANDS".

(1) ADDRESSABLE mode

The ADDRESSABLE mode is a normal mode. In this mode, the analyzer is controlled by an external controller.
If the built-in BASIC program of the analyzer is run in this mode, the analyzer will operate as follows:

① If "CONTROL 7;4" of the BASIC command has not been set:

Data can be transmitted/received between the built-in BASIC of the analyzer and an external controller.
However, since the ENTER and OUTPUT instructions of the built-in BASIC have higher priority, setting cannot be performed using a GPIB command from the external controller.
Perform setting using a GPIB command from the external controller, stop the built-in BASIC program or set "CONTROL 7;4".

② If "CONTROL 7;4" of the BASIC command has been set:

In contrast with ①, setting can be performed using a GPIB command from an external controller.
In other words, the system operates in the same manner as when the built-in BASIC is stopped. However, no data can be transmitted/received between the built-in BASIC and the external controller.

(2) SYSTEM CONTROLLER mode

The built-in BASIC program enables the analyzer to control the measurement function and the externally connected units.

2. OPERATING BASICS

2.1 Overview of Operation

The loading, executing and ending of the BASIC program are made by pressing the soft keys and entering the commands from the external keyboard. And it is also able to be input or executed programs by the external computer through GPIB.

2.2 Panel Operation

2.2.1 Inputting, Executing and Ending Program

Creating the BASIC program is executed by the programming with connecting the external keyboard to this analyzer or by saving the BASIC program in ASCII file created by the other personal computer to the memory cards.
To operate the loading, executing and ending the BASIC program that is already created, follow the steps below.

- Procedure

① Insert the memory card in which the BASIC program to execute is saved into the memory card drive of this analyzer.

② Press CNTRLR ON key under the panel display.
The soft keys used exclusively by the BASIC program are appeared in the display.

③ Select the device of the memory card that is inserted in step 1 by the Device of the soft keys. The active display is switched every time this soft key is pressed.In this active display, MA means memory card drive A, MB means memory card drive B.

④ Pressing the soft key LOAD makes the file selecting window displayed. Select the optional file name by the rotary knob or the step keys "↑, ↓".

CAUTION

The file selecting window is not be able to be appeared in the screen exclusive to BASIC. Press "Change Screen" key to make the screen in the waveforms condition and press the "LOAD" key.

⑤ The selected file is loaded by pressing the rotary knob or ENTER key.

⑥ The program is executed by pressing the soft key RUN.

⑦ The execution of the program is ended by pressing the STOP key.

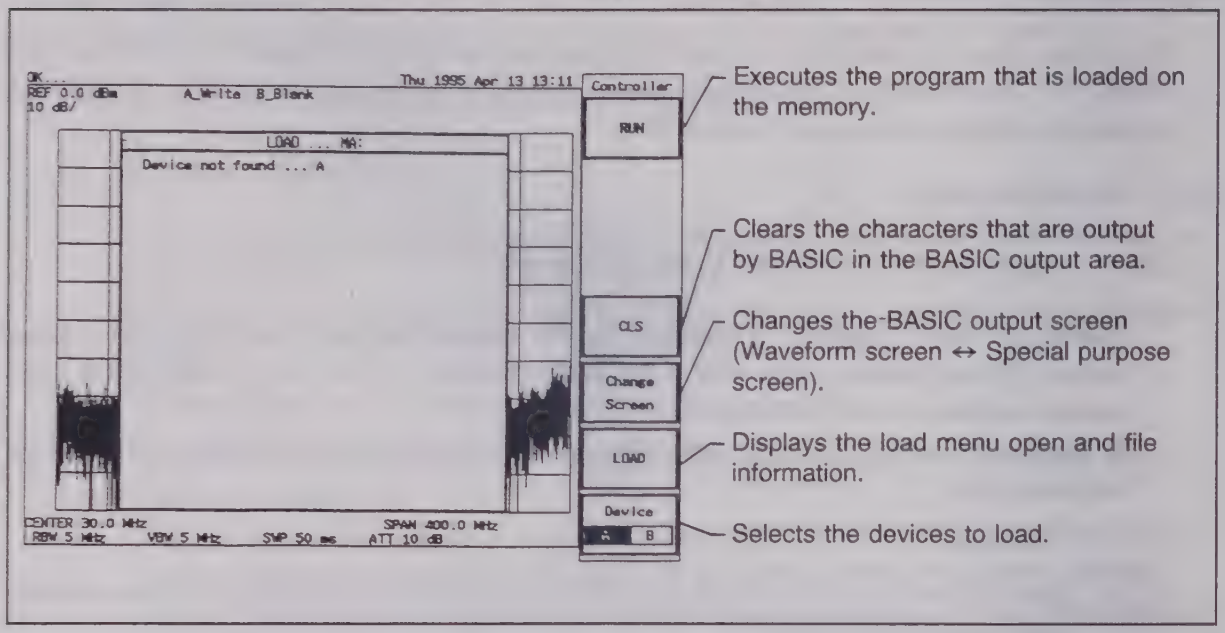


Figure 2-1 Loading and Executing of Program

2.2.2 Data Input Keys

Use the ten-key (0 to 9), the period key and back space key for the inputting on INPUT statement, and press ENTER key at last to complete the inputting.
It is possible to erase characters of the input data one by one by pressing the back space key, if it is before pressing the ENTER key.

2.2.3 Function Keys

The soft keys (1 to 7) are used for generating the key interrupt to the ON KEY ... statement.

2.3 Memory Card

In the memory card, the set condition, the BASIC program and the data files from inside of the BASIC program and so on are able to be recorded and regenerated.
The memory card format complies with MS-DOS, so it is possible to create a program and to analyze data on the MS-DOS-capable personal computer.
The following memory cards are able to be used in this analyzer.

2.3.1 Usable Memory Card

- Adapted to JEIDA Ver.4.0 or higher (68 pin two piece connector).
  TYPE1

- Only the following Memory types are permitted.
  Common memory : SRAM
  Attribute memory : Any one of the SRAM, EPROM, MASKROM, EEPROM, OTPROM or flash memory is all right.

- Formatting
  MS-DOS format.
  Corresponding to the various kinds of memory size.

CAUTION

Only the memory cards that are adapted to the PC card guide line Ver 4.0 of the Japan Electronic Industry Development Association (JEIDA) or to PCMCIA Release 2.0 or higher that is the United States of America standards are permitted. Use the memory cards only after making sure that those are adapted to the standards as above. For details, refer to "R3465 OPERATION MANUAL" or "R3272 OPERATION MANUAL".

2.3.2 Memory Card Specifications

Table 2-1 Memory Card Specifications

Specifications	Memory Card
Connector	68 Pin two Piece Connector
Interface	In accordance with JEIDA Ver.4.0
Dimensions	54 (Width) × 86 (Length) × 3.3 (Thickness) mm
Operating Environment	No condensation Operating environment: 0 to 55°C Storage environment: -20 to 60°C Relative humidity: Less than 95%
Write protect	Switching ON and OFF by the switch. It is impossible to write on the side of ON.

2.3.3 Note on Handling the Memory Card

- Keep dust out from the hole of the connector.
  It causes the defective contact or the damage of the connector.

- Do not touch the connector with a material like a metal needle and so on.
  It may causes the static electricity destruction.

- Do not bend it or give a great shock on it.

- Keep it away from water.

2.3.4 Insertion and Ejection of Memory Card

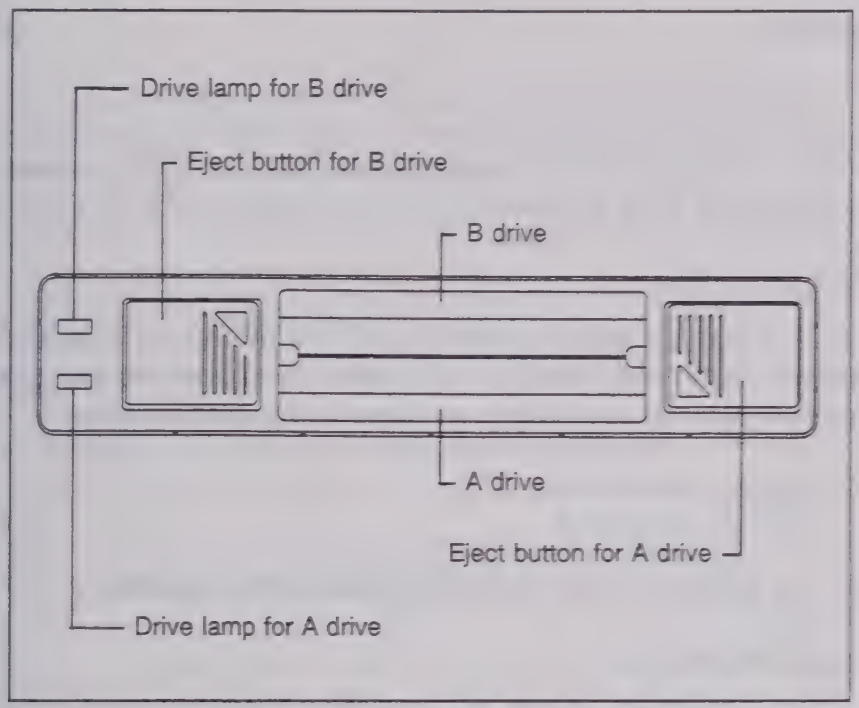


Figure 2-1 Drive Slot for Memory Card

The drive slots for the memory card are on the right upper of the front panel.

① Insert the memory card with the printed side up.

② The drive lamp is turned on yellow when the memory card is inserted.

③ When the memory card is ejected, press the eject button only after making sure that the drive lamp is turned on yellow.

CAUTION

The drive lamp is turned on red when the card is given access. Do not press the eject button to eject the memory card when the drive lamp is turned on red.
In the case that the memory card is ejected when the drive lamp is turned on red, the data in the memory card is not guaranteed.

2.4 File Management

2.4.1 Summary

(1) File

Generally, the information of a totality is called "file". The BASIC program edited on the personal computer or the data created by the internal BASIC and so on are all stored as files.

(2) Drive

The place to store the files is the memory card. The device used to read/write data from/to the memory card is called "drive" and the memory card can be managed by each drive. The following two drives are accessible by the internal BASIC of this analyzer.

MA : Memory Card Drive A
MB : Memory Card Drive B

Note The drive name, MA or MB, should be specified in capital letters.

(3) Storing and Recalling File

Specify any one drive and a file name to store or recall the file by BASIC.
Append a colon after the drive name in specifying the drive. The drive name can be omitted. If omitted, the current drive (the current directory) is selected automatically.

Format: Executing Command "[Drive:] <File Name>"

(4) Initializing Memory Card

The commands for initializing the memory card are not prepared in this BASIC. If you want to initialize a memory card, use Format Card A or Format Card B function in the software key menu of SAVE/RECALL in this analyzer. Refer the instruction manuals of each model for the further information.

2.4.2 File Management

(1) CHDIR

The current drive (the current directory) is changed with the CHDIR command.
This command cannot be used during executing the program. (However it can be used in the case that it is specified in the lines of the program.)

```
CHDIR
```

The current is changed to the home directory in the current drive.

```
CHDIR "ADVAN"
```

The current is changed to the directory ADVAN in the current drive.

```
CHDIR "MA:"
```

The current is changed to the current directory in the memory card A drive.

```
CHDIR "MA:/"
```

The current is changed to the home directory in the memory card A drive.

```
CHDIR "../"
```

The current is changed to the master directory of the current directory.

(2) CAT

The files of the memory cards in the drive and the directory information are displayed with the CAT command.

```
CAT
```

All in the current drive (the current directory) are displayed.

```
CAT "MA:."
```

All in the memory card A drive are displayed.

```
CAT "MA:*.BAS"
```

The files that are with extension ".BAS" in the memory card B drive are displayed.

```
CAT "../"
```

The inside of the directory that is one below the current directory is displayed.

2.4.3 Storing Files

With the SAVE command, the program that currently exists in the memory is stored into the device in the specified drive by giving an optional file name.
Take note that the data of the file is updated in the case that the same file name as that already been existing in the device is specified.

```
SAVE "AAA.BAS"
```

Stores the file in the current drive (the current directory).

```
SAVE "MA:BBB.BAS"
```

Stores the file in the memory card A drive.

2.4.4 Loading Files

The BASIC program files saved in the specified drive (directory) are loaded on the memory with the LOAD command.

```
LOAD "AAA.BAS"
```

Loads the file from the current drive (the current directory).

```
LOAD "MA:BBB.BAS"
```

Loads the file from the memory card A drive.

CAUTION

When the program created by the personal computer is loaded, if the characters except that are able to be managed in BASIC are detected in the program statements (except that they are enclosed in double quotation marks), message "Not available char (yyy)" is output and the loading is aborted.

2.4.5 Erasing Files

The unnecessary files are erased with the PURGE command.

```
PURGE "AAA.BAS"
```

The AAA. BAS file is erased from the current drive (the current directory).

```
PURGE "MA:BBB.BAS"
```

The BBB. BAS file is erased from the memory card A drive.

2.4.6 Changing File Name

The existing file names can be changed to the other file names with the RENAME command. Only the file names are changed but the data of the files are not changed.

```
RENAME "AAA.BAS", "BBB.BAS"
```

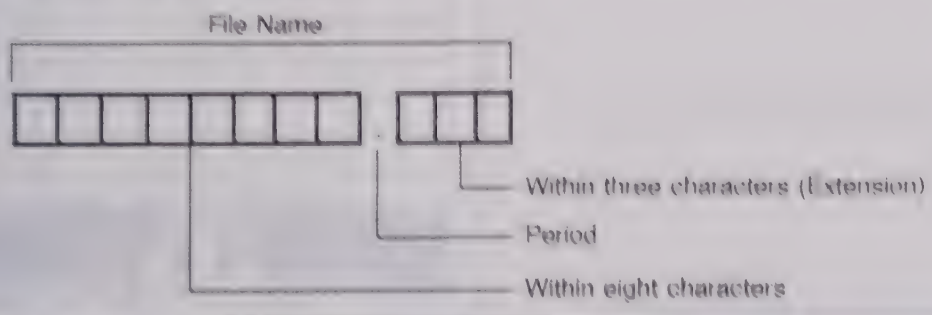
The file name AAA. BAS in the current drive (the current directory) is changed to the BBB. BAS.

```
RENAME "MA:AAA.BAS", "BBB.BAS"
```

The file name AAA. BAS in the memory card A drive is changed to BBB. BAS.

CAUTION

The file names should be written in alphanumeric characters and marks (except double quotation mark (")) as shown below.



2.5 Screen Layout

The screen layout of this analyzer when the internal BASIC is used is as follows.
The two kinds of screens are supported in using the internal BASIC of this analyzer. They are "BASIC Waveform Screen" used by composing with waveform screen (See Figure 2-3) and "Special Purpose BASIC Screen" displaying only BASIC waveform hiding the waveform screen (See Figure 2-4).
The two kinds of screens are switched with the Change Screen key of the soft key for the internal BASIC or "CONTROL 8;0" (BASIC Waveform Screen) and "CONTROL 8;1" (Special Purpose BASIC Screen) of BASIC command.

- BASIC Message Display Area:
  The special purpose area for the error messages and so on output by the internal BASIC.
- BASIC Output Area:
  The area in which the character strings output by executing a command in the internal BASIC are displayed
  The data output by executing the command LIST, CAT and so on are displayed in this area.
- Command Input Area:
  The area in which the key inputting from the keyboard when the external keyboard is connected to this analyzer or the characters corresponding to the input from the ten-key on the panel of this analyzer are displayed.
  It is used when the commands are input to the internal BASIC in using the external keyboard and so on.

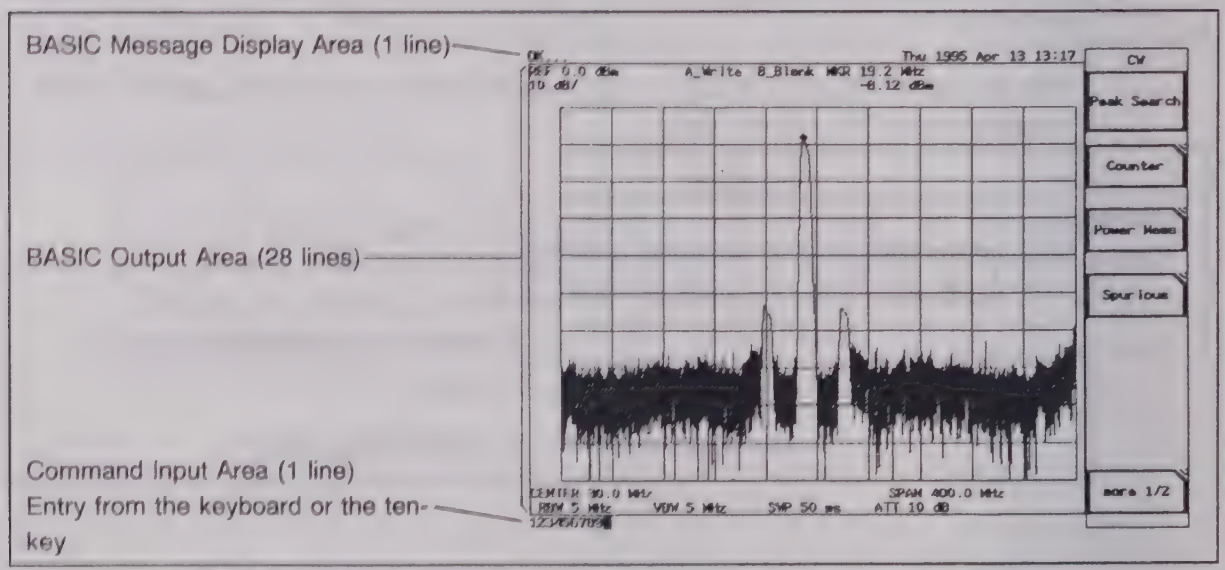


Figure 2-3 BASIC Waveform Screen (Waveform Screen + BASIC Screen)

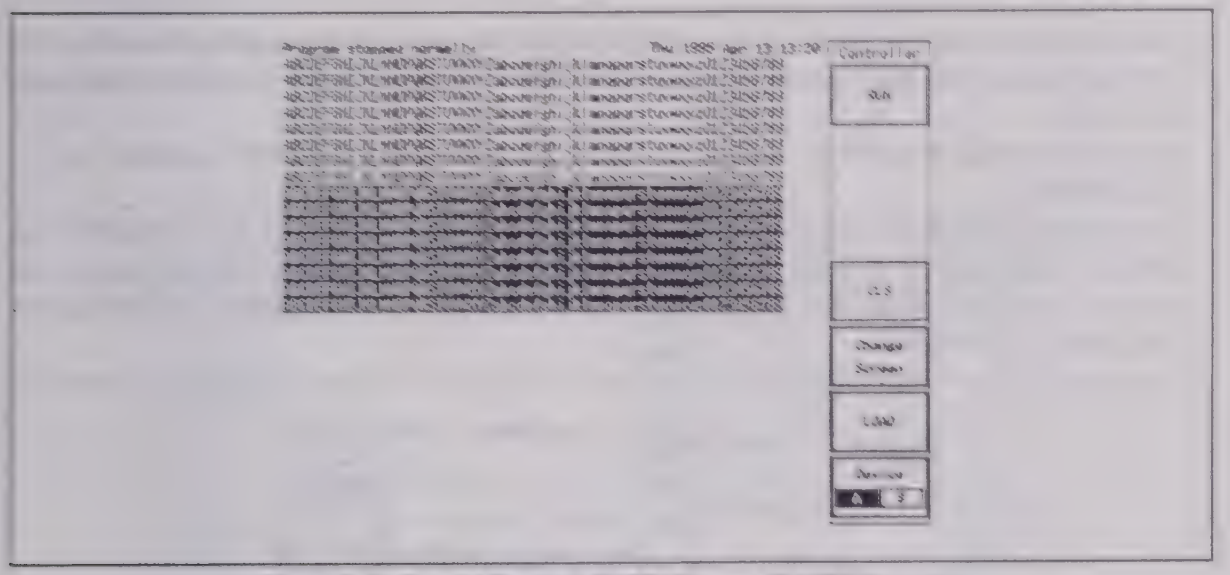


Figure 2-4 Special Purpose BASIC Screen (BASIC Screen)

2.6 External Keyboard

The external keyboard is necessary for editing a simple program on this analyzer. Connecting the external keyboard with this analyzer, it is possible to edit the loaded program, to modify or add data, by one line.

The usable keyboards are the 101 type or the 106 type equivalent keyboards (With connector form of mini DIN6 pin).

The changing between the 101 type and the 106 type can be made on the item Ext. Keyboard in the dialog box displayed by pressing the GPIB & Others soft key after pressing the LCL key of this analyzer. Select one of 101 key and 106 key with the rotary knob and determine by pressing the rotary knob or ENTER key.

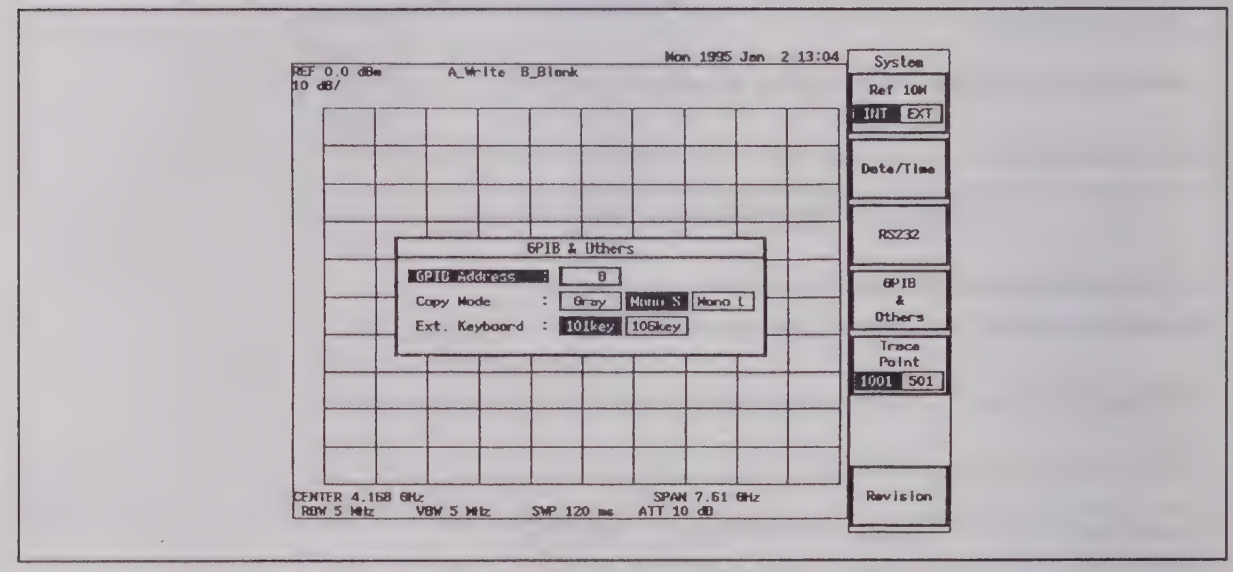
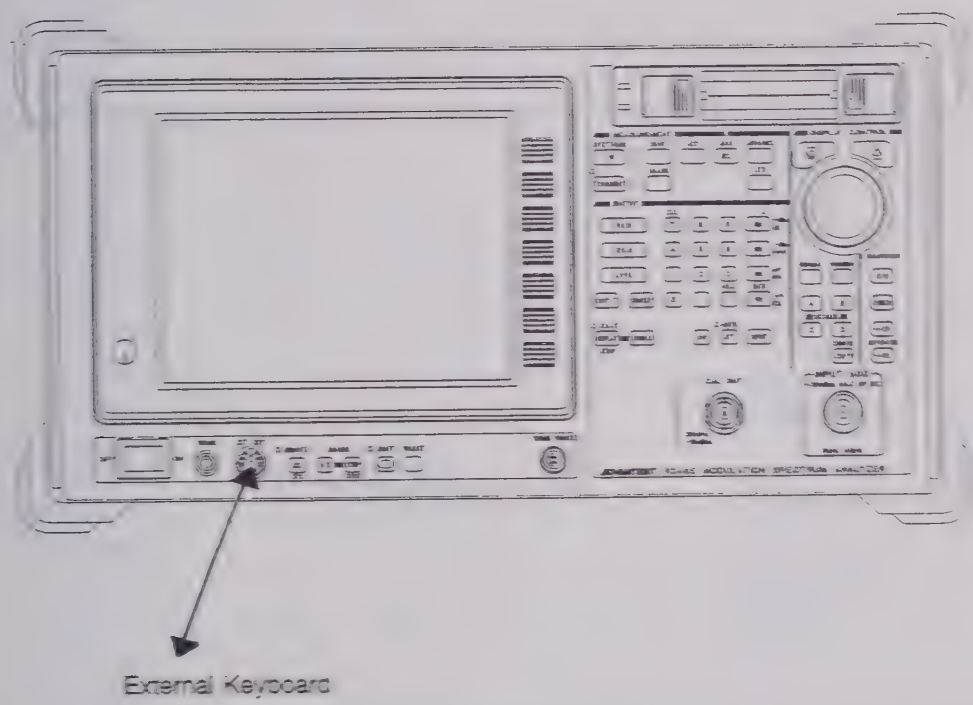


Figure 2-5 Changing between 101 Type and 106 Type (GPIB & Others)

2.6.1 Connecting External Keyboard

The external keyboard is connected to the EXT KEY connector of this analyzer.

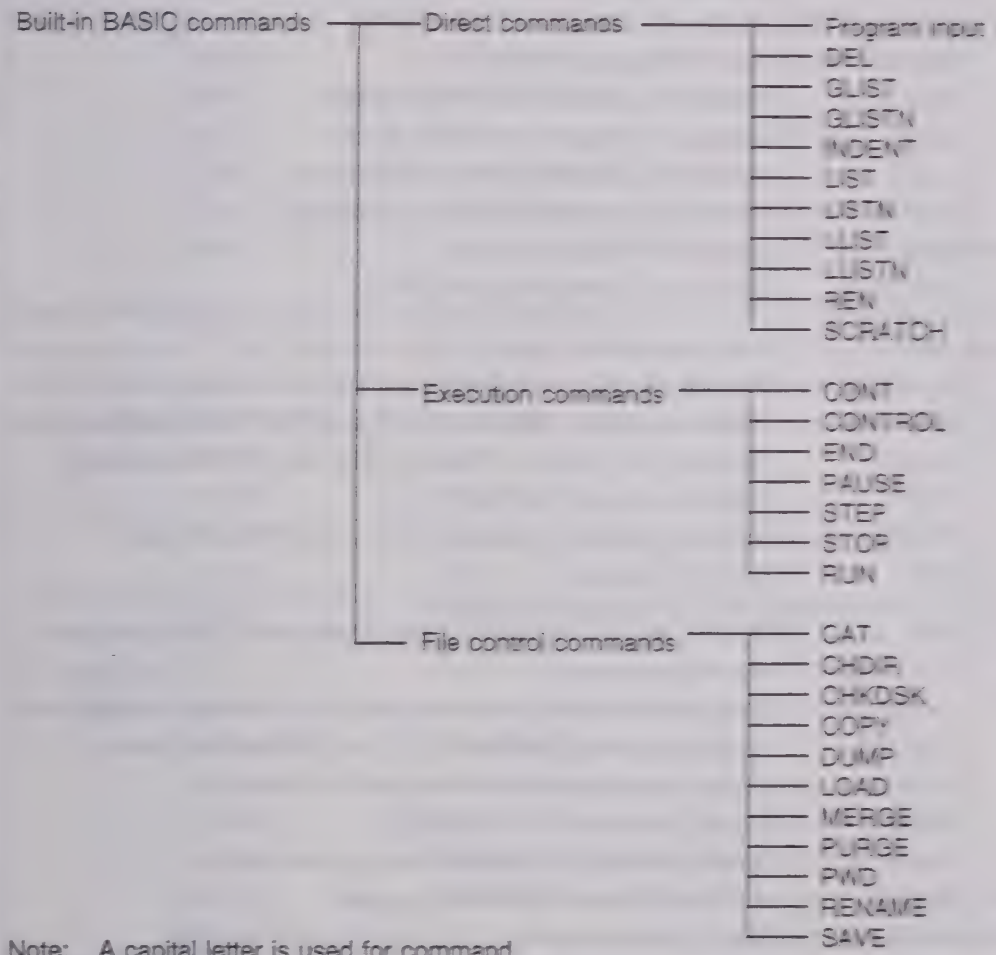


MEMO

3. BASIC COMMANDS

3.1 Various Commands

Built-in BASIC has commands to edit, carry out programs and operate files. The following shows the structure of the built-in BASIC commands.



Note: A capital letter is used for command.

3.1.1 List of Command Function

	Command	Function
EDIT commands	Program input	Stores the statement as a program.
	DEL	Deletes the specified line number.
	GLIST	Outputs the program list to the GPIB.
	GLISTN	Outputs the program list to the GPIB.
	INDENT	Indents the program list.
	LIST	Displays the program list on the screen.
	LISTN	Displays the program list on the screen.
	LLIST	Outputs the program list to the parallel port.
	LLISTN	Outputs the program list to the parallel port.
	REN	Changes the line number.
	SCRATCH	Deletes the already input program.
EXECUTION commands	CONT	Runs the program again.
	CONTROL	Sets the BASIC control variables. (Environment setup)
	END	Ends the program. (Disables CONT and STEP command)
	PAUSE	Suspends the program. (Enables CONT and STEP command)
	STEP	Runs the program one line.
	STOP	Stops the program. (Enables CONT and STEP command)
	RUN	Runs the program.
FILE control commands	CAT	Displays the file information stored in the media inside the drive onto the screen.
	CHDIR	Changes the current to the specified drive and the directory.
	CHKDSK	Displays the media information in the drive onto the screen.
	COPY	Copies the file recorded in the specified media.
	DUMP	Dumps the data of the file in ASCII.
	LOAD	Reads the BASIC program file onto the memory.
	MERGE	Merges and erases the BASIC program.
	PURGE	Purges the file recorded in the media inside the drive.
	PWD	Displays the path name of the current drive on the top of the line of the screen.
	RENAME	Renames the file name recorded in the specified media.
	SAVE	Saves (Stores) the program in the memory card.

3.1.2 List of Command Syntax

	Command	Syntax
EDIT commands	Program input	Line number Statement
	DEL	DEL <Start line> [, Last line]
	GLIST	GLIST [Start line] [, [Last line]]
	GLISTN	GLISTN [Start line] [, [Line number]]
	INDENT	INDENT <Start postion, increment >
	LIST	LIST [Start line] [, [Last line]]
	LISTN	LISTN [Start line] [, [Line number]]
	LLIST	LLIST [Start line] [, [Last line]]
	LLISTN	LLISTN [Start line] [, [Line number]]
	REN	REN [[Current line number] [, <New line number> [, <Increment>]]]
	SCRATCH	SCRATCH [1/2]
EXECUTION commands	CONT	CONT [Line number | Label expression]
	CONTROL	CONTROL <Resistor number>:<Value>
	END	END
	PAUSE	PAUSE
	STEP	STEP [Line number | Label expression]
	STOP	STOP
	RUN	RUN [Line number | Label expression]
FILE control commands	CAT	CAT ["[Drive name:] [Directory] [Wild card]"]
	CHDIR	CHDIR "[Drive name:] Directory name"
	CHKDSK	CHKDSK ["Drive name:"]
	COPY	COPY "Current file name", "New file name"
	DUMP	DUMP "[Drive name:] [Directory] File name"
	LOAD	LOAD "[Drive name:] [Directory] File name"
	MERGE	MERGE "[Drive name:] [Directory] File name", Merge start line number], [Start line | Labe expression"]
		MERGE DEL
	PURGE	PURGE "[Drive name:] [Directory] File name"
	PWD	PWD
	RENAME	RENAME "Current file name", "New file name"
	SAVE	SAVE "[Drive name:] [Directory] File name"

3.1.3 Precautions Common to All Commands

The following precautions are common to all of the built-in BASIC commands:

(1) Parameters

The character string representation expression and numeric value representation expression can be used to specify command parameters. In other words, variables used in the BASIC command can be used. If the number used is a real number, digits to the right of the decimal point will be omitted.
The description of each command uses representations such as integers and character strings for easy understanding.

(2) Boundary of expression

In principle, when the BASIC command uses multiple expressions continuously, a space can be used instead of a comma, as long as the boundary of the expressions can be interpreted in the syntax.

(3) Line number

The line number setting range is 1 to 65535.
If 0 or any value below the first line number of the program is specified, the analyzer will interpret that the first line of the program has been specified.
If 65535 or any value over the last line number of the program is specified, the analyzer will interpret that the last line of the program has been specified.

3.2 Command Grammar and Application

1. Program Input

The commands and statements described in Chapters 3 and 4 can be entered as a program if line numbers are added to them.
If the same line number exists in a program which has already been input, the newly entered number will replace it. If the same line number does not exist, the new number will be added or inserted.

2. CAT

Outline

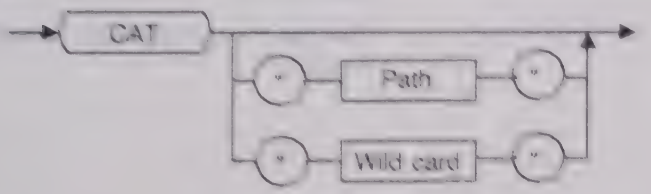
When a drive (directory) name is specified, the file data saved in the media on the drive (directory) is displayed onto the screen by the CAT command. If not specified, however, the file data saved in the media on the current drive (current directory) is displayed.

The file information to be output is the registered number (the variable), the file name (the directory name), the number of bytes used, the file attribute, the date the file was created in that order from the left. When the insertion and extraction is detected, the current directory is moved to the root directory.

File Attribute	1: READ ONLY 16: DIRECTORY 32: ARCHIVE FILE

Syntax

(1)-1



(1)-2
CAT ["[Drive name:] [Directory/] [Wild card]"]

Description

CAT: Displays all of the inside of the current drive (the current directory).

CAT "MA:": Displays all of the inside the memory card A drive.

CAT "MB:*.BAS": Displays the file with the extension'.BAS' inside the memory card B drive.

CAT "../": Displays the inside of the master directory of the current directory.

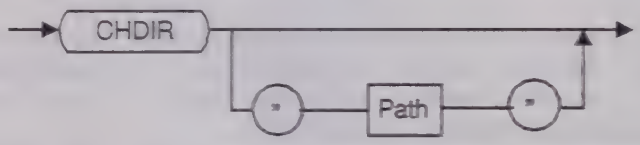
3. CHDIR

Outline

The CHDIR command is used to change the current to the specified drive and directory.
The path name of the new current drive (the current directory) is displayed on the top line on the screen. the current path name changed by CHDIR can be verified with the PWD statement (Refer to "21. PWD" in this section).
This command cannot be used during the execution of a program. (However it is able to be written in the program.)

Syntax

(1)-1



(1)-2

CHDIR "[Drive name:/] Directory | ../[../../..]"

Description

CHDIR:	Changes the current to the home directory of the current drive.
CHDIR "ADVAN":	Changes the current to the directory ADVAN of the current drive.
CHDIR "MA:":	Changes the current to the current directory in the memory card A drive.
CHDIR "MA:":	Changes the current to the home directory in the memory card A drive.
CHDIR "..":	Changes the current to the master directory of the current directory.

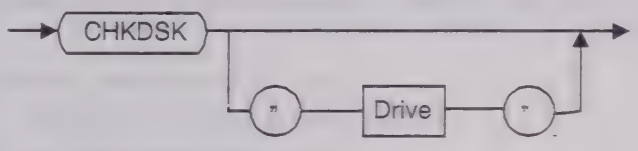
4. CHKDSK

Outline

When a drive name is specified, the media data in the drive is displayed onto the screen by the CHKDSK command. If not specified, however, the media data in the current drive is displayed.

Syntax

(1)-1



(1)-2

CHKDSK [" Drive name: "]

Description

CHKDSK: Displays the media data in the current drive.

CHKDSK "MA:": Displays the media data in the memory card drive A.

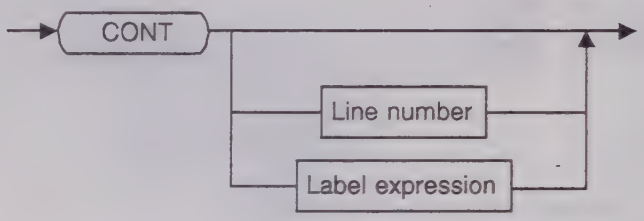
5. CONT

Outline

The CONT command is used to restart the BASIC program halted with PAUSE (refer "18. PAUSE" in this section).

Syntax

(1)-1



(1)-2

CONT [Line number | Label expression]

Description

CONT: Restarts the BASIC program at the next of the line where the program was halted.

CONT 100: Restarts the program at the 100th line.

CONT *Lbl: Restarts the program at the *Lb1 line.

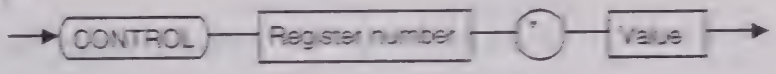
6. CONTROL

Outline

The CONTROL command is used to set the detailed values concerning the built-in BASIC control (environment variable).

Syntax

(1)-1



(1)-2

CONTROL register number : value

Description

<Register 1> ··· Initial value: 79 (Unassigned register at present)

Sets a serial I/O port. The sum of the following values is used to specify the serial I/O port. The underlined-value is each default value which has been already set when the analyzer is turned on.

- Baud rate:
  0; 1200 baud, 1; 2400 baud, 2; 4800 baud, 3; 9600 baud
- Character length:
  0; 5 bits, 4; 6 bits, 8; 7 bits, 12; 8 bits
- Parity:
  0; None, 16; Odd, 48; Even
- Stop-bit number:
  0; None, 64; 1 bit, 128; 1 1/2 bits, 192; 2 bits

Example: When 9600 baud for baud rate, 8 bits for character length, even parity for parity, and 2 bits for stop-bit number are used:

CONTROL 1;3 + 12 + 48 + 192
or CONTROL 1;255

<Register 2> ··· Initial value: 0

With the command LIST, LLISTN, GLIST or GLISTN, specify the print start position from the left side by entering the number of the space.

Example: When the list output is moved to the right by five characters:

CONTROL 2;5

<Register 3> ··· Initial value: 0

Specifies whether the BASIC command will be displayed in full name or short name.

0: Full name
1: Short name

<Register 4> ··· Initial value: 1

Specifies whether the variables, labels and so on are in all-caps representation or in initial-caps representation when the program list is executed.

0: All-caps representation
1: Initial-caps representation (The commands and data after reservation are in all-caps representation.)

<Register 5> ··· Initial value: 0

Sets whether the cursor is displayed on the screen or not.

0: The cursor is not displayed.
1: The cursor is displayed.

<Register 7> ··· Initial value: 0

Used for GPIB setting. Each value are able to be set as follows.

0: sets GPIB mode to ADDRESSABLE.
1 : Sets GPIB mode to SYSTEM CONTROLLER.
2 : Transits REQUEST CONTROL (request for control privilege).
4 : Enables GPIB command setting from the external controller during BASIC operation.

<Register 8> ··· Initial value: 0

Switches the display on the screen.

0: The waveform screen (The output characters are merged into the normal waveform display.)
1: BASIC screen (The waveform is not displayed.)

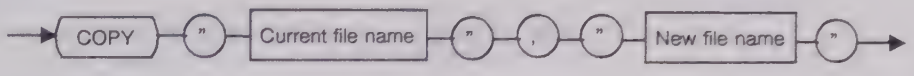
7. COPY

Outline

The COPY command is used to copy the file stored in the specified media.

Syntax

(1)-1



(1)-2

COPY "current file name", "new file name"

Description

- The COPY command copies the contents of the current file name to a new file name.
- When a new file name has already existed, the contents of the current file is overwritten.
- Both of two file names can be specified by using a character-string expression.

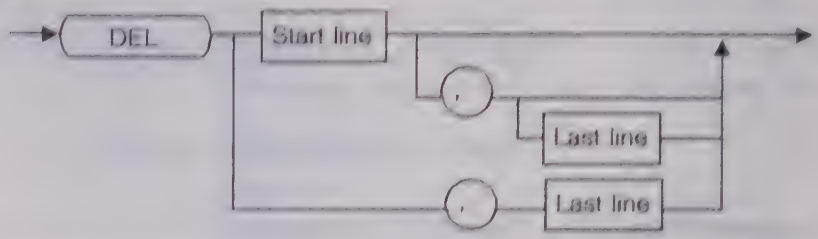
8. DEL

Outline

The DEL command is used to delete lines in the program.

Syntax

(1)-1



(1)-2

DEL ＜Start line [, [last line] ＞ | ＜, last line＞

Note: A space may be used instead of a comma.
The line number setting range is 1 through 65535.

Description

- The DEL command deletes the program from the start line to the last line.
- If the line number is omitted, the no operation will be performed.

Example

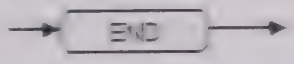
DEL 10	Deletes the 10th line only of the program.
DEL 10,	Deletes the program from line 10 to the last line.
DEL 10,100	Deletes the program from line 10 to line 100.
DEL , 100	Deletes the program from the start line to line 100.

9. END

Outline The END command is used to end the BASIC program.

Syntax

(1)-1



(1)-2

END

Description Terminates the BASIC program, or makes the BASIC program ended (set

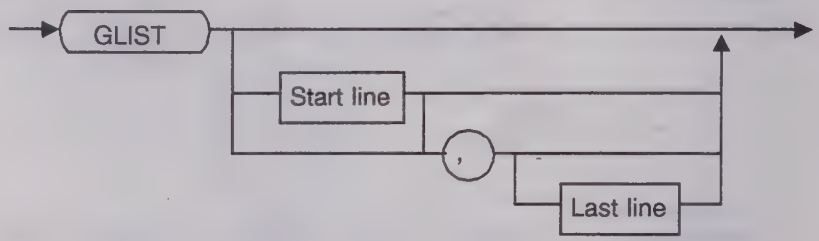
10. GLIST

Outline

The GLIST command is used to output a program list to peripheral devices such as a printer, etc. through the GPIB.

Syntax

(1)-1



(1)-2

GLIST [Start line [, [last line]]] | [, [last line]]

Note: A space may be used instead of a comma.
The line number setting range is 1 through 65535.

Description

- The GLIST command outputs the BASIC programs list to peripheral devices such as a printer, etc. connected with the GPIB.
- The printer GPIB address can be define by the PRINTER statement or the panel key operation of R3465 Series.

Example

GLIST	Outputs all lines of the program list.
GLIST 100	Outputs the 100th line only of the program list.
GLIST 100,	Outputs the program list from line 100 to the last line.
GLIST 100, 200	Outputs the program list from line 100 to line 200.
GLIST ,	Outputs all lines of the program list. (Same as GLIST)
GLIST , 200	Outputs the program list from the start line to line 200.

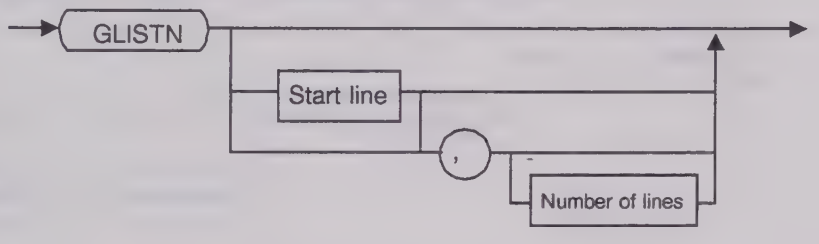
11. GLISTN

Outline

The GLISTN command is used to output a program list to peripheral devices such as a printer, etc. through the GPIB.

Syntax

(1)-1



(1)-2

GLISTN [Start line [, [number of lines]]] | [, [number of lines]]

Note: A space may be used instead of a comma.
The line number setting range is 1 through 65535.

Description

- The GLISTN command outputs the BASIC programs list to peripheral devices such as a printer, etc. connected with the GPIB.
- The printer GPIB address can be define by the PRINTER statement or the panel key operation of R3465 Series.
- The GLISTN command outputs specified lines of the program list from the start line number specified at the start line.
- When the line number is a negative value, this command outputs the program list toward the lower order numbers.

Example

GLISTN	Outputs all lines of the program list.
GLISTN 100	Outputs the 100th line only of the program list.
GLISTN 100,	Outputs the program list from line 100 to the last line.
GLISTN 100, 20	Outputs 20 lines of the program list from line 100.
GLISTN ,	Outputs all lines of the program list. (Same as GLISTN)
GLISTN , 20	Outputs 20 lines of the program list from the start line.

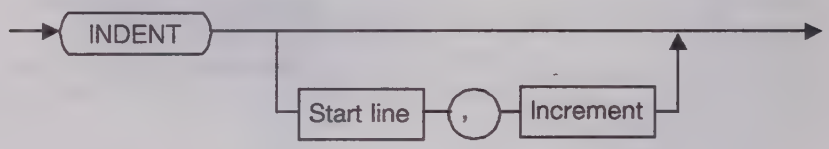
12. INDENT

Outline

The INDENT command is used to reformat the program list on the memory to reflect the construction of the syntax. The start position is at the first character of the first statement after each line number. The increased number of digits are the number of the characters with which the first character moves from side to side when the level of the nest of the program is changed.

Syntax

(1)-1



(1)-2

INDENT [Start position, Increment]

Available range

Start position: 1 to 32

Increment: 0 to 10

Description

- INDENT → Indents the increased two characters of the digits at the sixth character from the start position.
- INDENT 8, 3 → Indents the increased three characters of the digits at the eighth character from the start position.

Caution

The INDENT command is used to indent in one of the statements of IF (ELSE, ELSE IF), FOR and SELECT (CASE, CASE ELSE). But only on the simple IF sentence, the indenting is not executed depending on the syntax.
The END IF, NEXT and END SELECT statements invert the indentation.

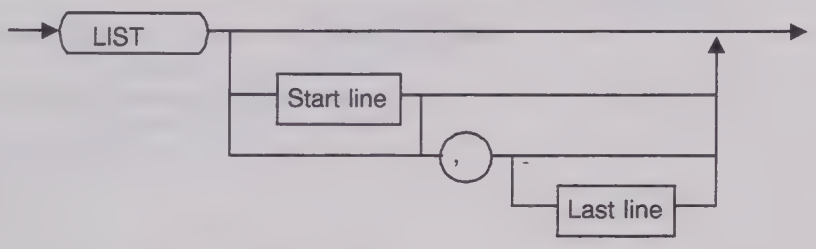
13. LIST

Outline

The LIST command is used to display a program list on the display.

Syntax

(1)-1



(1)-2

LIST [Start line [, [last line]]] | [, [last line]]

Note: A space may be used instead of a comma.
The line number setting range is 1 through 65535.

Description

- The LIST command displays the BASIC program list specified by the parameters on the display.
- The display of the program list can be aborted using the STOP key. However, since the stop operation differs from the program operation, the program list cannot be redisplayed from the aborted line.

Example

LIST	Outputs all lines of the program list.
LIST 100	Outputs the 100th line only of the program list.
LIST 100,	Outputs the program list from line 100 to the last line.
LIST 100, 200	Outputs the program list from line 100 to line 200.
LIST ,	Outputs all lines of the program list. (Same as LIST)
LIST , 200	Outputs the program list from the start line to line 200.

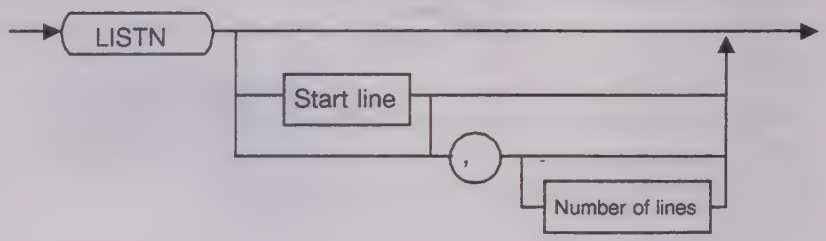
14. LISTN

Outline

The LISTN command is used to display a program list on the display.

Syntax

(1)-1



(1)-2

LISTN [Start line [, [number of lines]]] | [, [number of lines]]

Note: A space may be used instead of a comma.
The line number setting range is 1 through 65535.

Description

- The LISTN command displays the BASIC program list specified by the parameters on the display.

Example

LISTN	Outputs all lines of the program list.
LISTN 100	Outputs the 100th line only of the program list.
LISTN 100,	Outputs the program list from line 100 to the last line.
LISTN 100, 20	Outputs 20 lines of the program list from line 100.
LISTN ,	Outputs all lines of the program list. (Same as LISTN)
LISTN , 20	Outputs 20 lines of the program list from the start line.

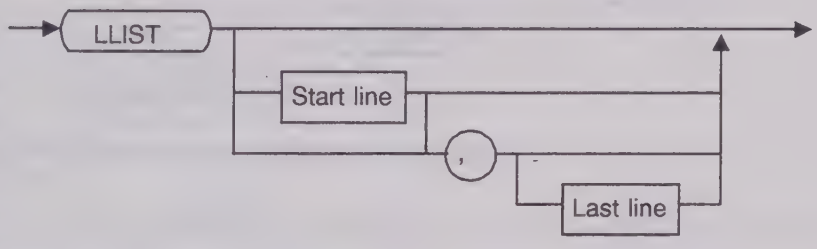
15. LLIST

Outline

The LLIST command is used to output a program list to peripheral devices such as a printer, etc. through the parallel port.

Syntax

(1)-1



(1)-2

LLIST [Start line [, [last line]]] | [, [last line]]

Note: A space may be used instead of a comma.
The line number setting range is 1 through 65535.

Description

- The LLIST command outputs the BASIC program list to peripheral devices such as a printer, etc. connected with the parallel port.

Example

LLIST	Outputs all lines of the program list.
LLIST 100	Outputs the 100th line only of the program list.
LLIST 100,	Outputs the program list from line 100 to the last line.
LLIST 100, 200	Outputs the program list from line 100 to line 200.
LLIST ,	Outputs all lines of the program list. (Same as LLIST)
LLIST , 200	Outputs the program list from the start line to line 200.

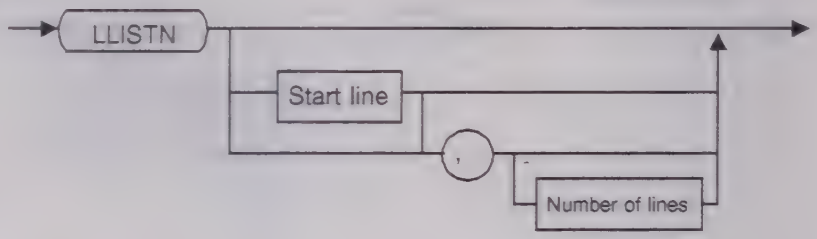
16. LLISTN

Outline

The LLISTN command is used to output a program list to peripheral devices such as a printer, etc through the parallel port.

Syntax

(1)-1



(1)-2

LLISTN [Start line [, [number of lines]]] | [, [number of lines]]

Note: The line number setting range is 1 through 65535.

Description

- The LLISTN command outputs the BASIC program list to peripheral devices such as a printer, etc. connected with the parallel port.
- The LLISTN command outputs specified lines of the program list from the start line number specified at the start line.
- When the line number is a negative value, this command outputs the program list toward the lower order line numbers.

Example

LLISTN	Outputs all lines of the program list.
LLISTN 100	Outputs the 100th line only of the program list.
LLISTN 100,	Outputs the program list from line 100 to the last line.
LLISTN 100, 20	Outputs 20 lines of the program list from line 100.
LLISTN ,	Outputs all lines of the program list. (Same as LLISTN)
LLISTN , 20	Outputs 20 lines of the program list from the start line.

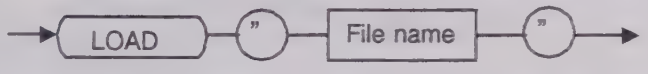
17. LOAD

Outline

The LOAD command is used to load the file saved in the drive.

Syntax

(1)-1



(1)-2

LOAD "[Drive name:] [Directory/] File name"

Description

- Loads the file specified by the file name. The files except BASIC cannot be loaded.

Examples

LOAD "AAA. BAS": Loads the data of AAA. BAS file stored in the media inside the current drive (the current directory).

LOAD "MA:BBB. BAS: Loads the data of BBB. BAS file stored in the memory card inside the memory card A drive.

Note: For the information how to handle files, refer to "2.4 File Management".

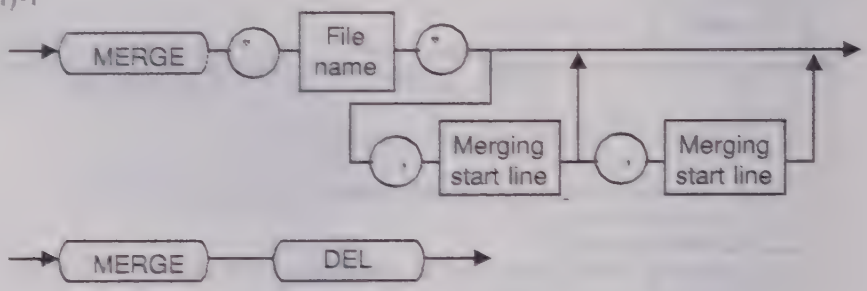
18. MERGE

Outline

The MERGE command is used to load and merge the files stored in the drive.And it is used to erase the merged program from the memory.

Syntax

(1)-1



(1)-2

MERGE "[Drive name:] [Directory/] File name",
[Merging start line number] [, Line number | Label expression]

Description

When a drive name is specified in the program on the memory, the specified file saved in the media on the drive is merged by the MERGE command. If not specified, however, the specified file saved in the media on the current drive is merged.

Adopts the line loaded from the media if there is the same line number in the program on the memory and in the program loaded (merged) from the media.

The line of which line number is not matched will be added or inserted to the program on the memory.

And only in the case that the program file without line number is merged, the merging start line number of the second arguments becomes enable. If the merging start line is specified, from the merging start line, and if it is not specified, from the last plus one line of the program existing in the memory, the line numbers are assigned in increments on one automatically.

The line number - label expression of the third argument specifies the execution start line number or the label expression after merging the program. When this argument is omitted or when the line number that is not existed in the merged program is specified, the execution start position becomes the first line of the merged program automatically.

And when the program line already merged is to be erased, spells out DEL continued to the MERGE command. The DEL command is not able to be used to delete the merged program. With executing this command, the part of the merged program is erased from the program on the memory, and the execution start position (line) of the program moves to the line next to the line where the MERGE command is executed.

The MERGE command differs from the LOAD command, since the BASIC buffer is not initialized before loading.

The combination of the SCRATCH and MERGE commands represents the same function as the LOAD command.

Examples

MERGE "AAA. BAS" :
Loads the data of the AAA. BAS file stored in the media inside the current drive (the current directory).

MERGE "MA:BBB. BAS" :
Loads the data of the BBB.BAS file stored in the memory card inside the memory card A drive.

MERGE "BBB",1000 :
Merges the program of the BBB file from the line number 1000 to the program on the memory, and after that, continues the execution of the program from the first line of the merged lines.

MERGE "BBB",100,200 :
Merges the program of the BBB file from the line number 100 to the program on the memory, and after that, continues the execution of the program from the 200th line of the merged lines.

MERGE "BBB",,*Lb1 :
Merges the program of the BBB file to the program on the memory, and after that, continues the execution of the program from the line number *Lb1 of the merged lines.

MERGE DEL :
Erases the lines of the program already merged.

Description

- When the program on the memory is the protected file, the lines of the program is not able to be merged within the last line number of this file. In this case, the following message is displayed.
(Cannot execute protected file)
- It is impossible to specify the label on the start line of the merging.
- It becomes effective that the specification of the line to start merging only when the program to be merged has no line number. When the merged program has the line numbers, it is merged as it is.
- When the execution start line is omitted or the specified line number is the number other than the merged program line, it continues the execution from the first line of the merged program.
- When the program that is to be merged by the MERGE command updates the line next to the line on the memory on which the MERGE command is executed, the program line is erased on executing the MERGE DEL command. Therefore, it does not continue that the execution of the program.

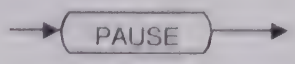
19. PAUSE

Outline

The PAUSE command is used to pause (suspend) a program operation.

Syntax

(1)-1



(1)-2

PAUSE

Description

- The PAUSE command suspends the BASIC program temporally, or the BASIC program itself stops the program temporally.
- The program is restarted again at the next line of the suspended line by the CONT command.

Example

```
10  FOR I=1 TO 9
20    GOTO 60
30    GOTO *PRT
40  NEXT
50  PAUSE
60  !
70  X = I * I
80  GOTO 30
90  *PRT
100 PRINT I; "*" ;I; "=" ;X
110 GOTO 40
```

20. PRINTER

Refer to "55. PRINTER" in section 4.3.

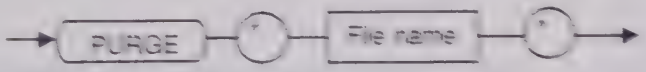
21. PURGE

Outline

When a drive name is specified, the specified file saved in the media on the drive is purged by the PURGE command. If not specified, however, the specified file saved in the media on the current drive is purged.

Syntax

(1)-1



(1)-2

PURGE "[Drive name:] [Directory name:] File name"

Description

- The PURGE command is used to purge the unwanted existing-files.

Examples

PURGE "AAA. BAS" : Purges the data of the AAA. BAS file stored in the current drive (the current directory).

PURGE "MA:BBB. BAS" : Purges the BBB. BAS file stored in the memory card inside the memory card A drive.

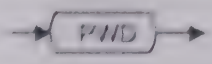
22. PWD

Outline

The PWD command is used to display the path name of the current drive (the current directory) onto the first line of the screen.

Syntax

(1)-1



(1)-2

PWD

Description

PWD : Displays the path name of the current drive (the current directory) onto the first line of the screen.

- The PWD command is used to display the path name at the position of the current that is to be changed by the CHDIR statement. (See "3. CHDIR" in this section.)

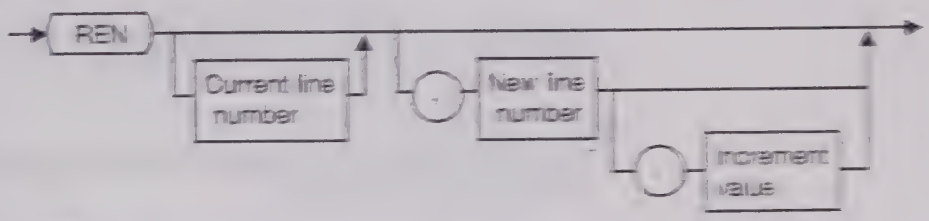
23. REN

Outline

The REN command is used to renew the line numbers of the program. This command is disabled during the execution of the program.

Syntax

(1)-1



(1)-2

REN [,[Current line number] [,New line number [,Increment value]]]

Note: A space may be used instead of a comma.
The setting range of the current line number, the new line number and the increment value is 1 through 65535.

Description

- The current line number specifies the head of the line number to be renewed in the current program.
- The new line number specifies the start of the renewed line number.
- The increment value specifies the step of the renewed line number.
- The REN command renews the line number used in the GOTO and GOSUB statements corresponding to the new line number.
- The REN command cannot be used to specify the line number exceeds 65535. Do not specify the program line with changing/modifying the order.

Example

REN : Renews the start line to 10, and changes the line number by 10 steps till the last line.

REN 50,100 : Renews the line number of 50 to 100, and after that, changes the line number by 10 steps till the last line.

REN 10,50,5 : Renews the line number of 10 to 50, and after that, changes the line number by 5 steps till the last line.

24. RENAME

Outline

The RENAME command is used to rename the file stored in the specified media.

Syntax

(1)-1

(1)-2

RENAME "[Drive name:] current file name", [Drive name:] "new file name"

Description

- The RENAME command renames only the file name stored without changing its contents.
- If the same file exists in a memory card which has already been created, then no operation will be performed.

Example

RENAME "AAA.BAS", "BBB.BAS" :
Renames the file "AAA. BAS" stored in the media of the current drive (the current direction) to "BBB. BAS".

RENAME "MA:AAA.BAS", "BBB.BAS" :
Renames the file "AAA. BAS" stored in the media of the memory card A drive to "BBB. BAS".

Note: For the information how to handle files, refer to "2. 4 File Management".

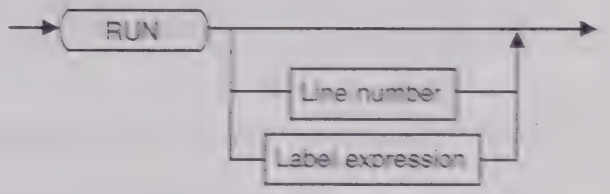
25. RUN

Outline

The RUN command is used to execute the BASIC program on the memory. And it is used to load the program from the media inside the drive and execute that program.

Syntax

(1)-1



(1)-2

RUN [line number | label expression | " [drive name:] [directory name] file name"]

Description

- The RUN command executes the BASIC program from the specified line.
- If no line number is specified, the program will be executed from the start line.
- When the RUN command is executed, all the variables are cleared and also the array declarations are forcibly cleared before program execution.

Example

RUN :	Executes the program on the memory at present.
RUN 100 :	Executes the program on the memory at present from the line number 100.
RUN "Lb1 :	Executes the program on the memory at present from the "Lb1 line.
RUN "AAA. BAS" :	Loads the AAA. BAS file stored in the media of the current drive (the current directory) and after that executes it.
RUN "MA:BBB. BAS" :	Loads the BBB. BAS file stored in the memory card inside the memory card A drive and after that executes it.

26. SAVE

Outline

When a drive name is specified, the specified file saved in the media on the drive is saved by the SAVE command. If not specified, however, the specified file saved in the media on the current drive is saved.

Syntax

(1)-1

(1)-2

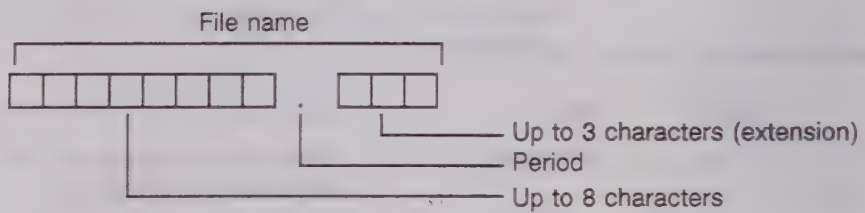
SAVE "[drive name:] [directory name/] file name"

Description

- The SAVE command stores the program (stored in the memory) into the file specified in the statement.
- If the already existed file name is specified, the specified file is assumed to update, then the file is overwritten.

CAUTION

The file name uses numerics, alphabets and symbols (except for double quotations), and specify the file name as follows:



Example

SAVE "AAA. BAS" : Saves the data of the AAA. BAS file stored in the media in the current drive (the current directory).

SAVE "MA:BBB. BAS": Saves the data of the BBB. BAS file stored in the memory card inside the memory card A drive.

Note: For the information how to handle files, refer to "2.4 File Management".

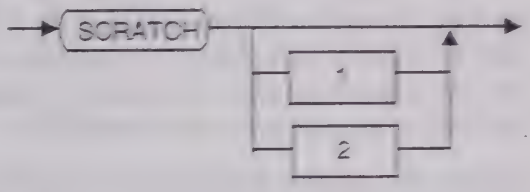
27. SCRATCH

Outline

The SCRATCH command is used to scratch (erase) the program on the memory.
This command is not able to be used during the execution of the program.

Syntax

(1)-1



(1)-2

SCRATCH [1 | 2]

Example

SCRATCH: Erases the program on the memory at present. (Data, Procedure)

SCRATCH 1: Initializes only the program data (the variable allocation values and so on). (Data)

SCRATCH 2: Erases the program list on the memory at present. (Procedure)

28. STEP

Outline

The STEP command is used to execute the only one line of the program on the memory. When the line number is specified, it executes that line. And when the line number is not specified, it executes the line next of the line suspended just before.

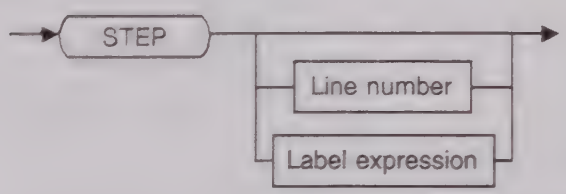
With the PAUSE statement in the program on the RUN condition of the program, or from the pause condition of the program by inputting "PAUSE" from the command line, the STEP command is used to execute the STEP and after that, it is possible to continue the execution of the program with the CONT command from the optional line.

And it is also possible to execute the step when the program is not on the RUN condition. However, in this case, specify definitely the STEP start line (the start label) and the executing start line just for the first time of the execution of the STEP command. After that, it is possible to execute one by one line with the ordinary STEP command.

And in this case, it is impossible to continue the execution from the present executing line by the CONT command.

Syntax

(1)-1



(1)-2

STEP [line number | label expression]

Description

STEP : Executes the next of the line suspended just before.

STEP 100 : Executes the one line with the line number 100.

STEP *Lb1 : Executes the one line with the line number *Lb1.

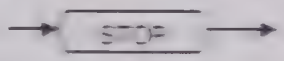
29. STOP

Outline

The STOP command is used to stop the BASIC program.

Syntax

[1-1]



[1-2]

STOP

Description

- The STOP command stops the BASIC program execution or the BASIC program itself stops the program execution.
- It is possible to continue the execution of the program from the stopped line with the CONT command or the STEP command.

MEMO

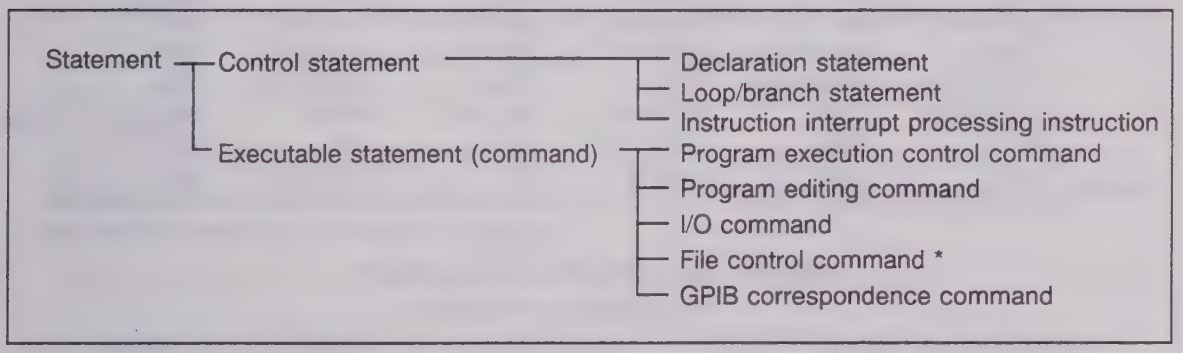
4. BASIC STATEMENT

4.1 Programming Rules

4.1.1 Program Structure

(1) Statement

The BASIC program consists of various statements.
The statements are grouped into two types; control statement and executable statement (command).
Each statement consists of key words and expressions. The decision of the construction is the syntax rule for grammar.



(2) Key word

The term whose meaning and application are predetermined with BASIC is called a "key word". The same name as the key word cannot be used for any other purpose.
The key word that is frequently used and whose full name is long has a short name.
To change the appearance from the full name to the short name, CONTROL command should be used to set the control register 3 should be set to "0".

For information of key word list, refer to Table 4-1.
The relationship between the full and short names is shown in Table 4-2.

Table 4-1 Key Word List

AND	AS	ASCII	USING	WAIT	XOR
BOR	BREAK	BXOR	BAND	BINARY	BNOT
CLS	CMD	COLOR	CASE	CLEAR	CLOSE
COUNT	CSR	COMMON	CONSOLE	CONTINUE	COS
DATE	DELAY	CSRLIN	CSRPOS	CURSOR	DATA
ELSE	ENABLE	DELIMITER	DIM	DISABLE	DSTAT
ERRM	ERRN	END	ENT	ENTER	ENTERF
FORMAT	FRE	ERROR	EVENT	EXP	FOR
GPRINT	IF	GLIST	GLISTN	GOSUB	GOTO
INTERFACE	INTR	INKEY	INP	INPUT	INTEGER
LINE	LISTEN	ISRQ	KEY	LABEL	LEN
LOCKOUT	LOG	LISTN	LLIST	LLISTN	LOCAL
OFF	ON	LPRINT	NEXT	NOT	NUM
OUTPUTF	PI	OPEN	OR	OUTPUT	OUT
PRINTF	READ	POS	PRINT	PRINTER	PRF
RESTORE	REQUEST	REAL	REM	REMOTE	REN
SPRINTF	SQR	RETURN	SELECT	SEND	SIN
THEN	TIME	SRQ	TALK	TAN	TEXT
UNT	USE	TIMER	TO	TRIGGER	UNL

Table 4-2 Correspondence Table between Full Name and Short Name

Full Name	Short Name
COMMON	COM
CURSOR	CSR
ENTER	ENT
INPUT	INP
OUTPUT	OUT
PRINTF	PRF
SPRINTF	SPRF
USING	USE
PRINT	?

(3) Expression

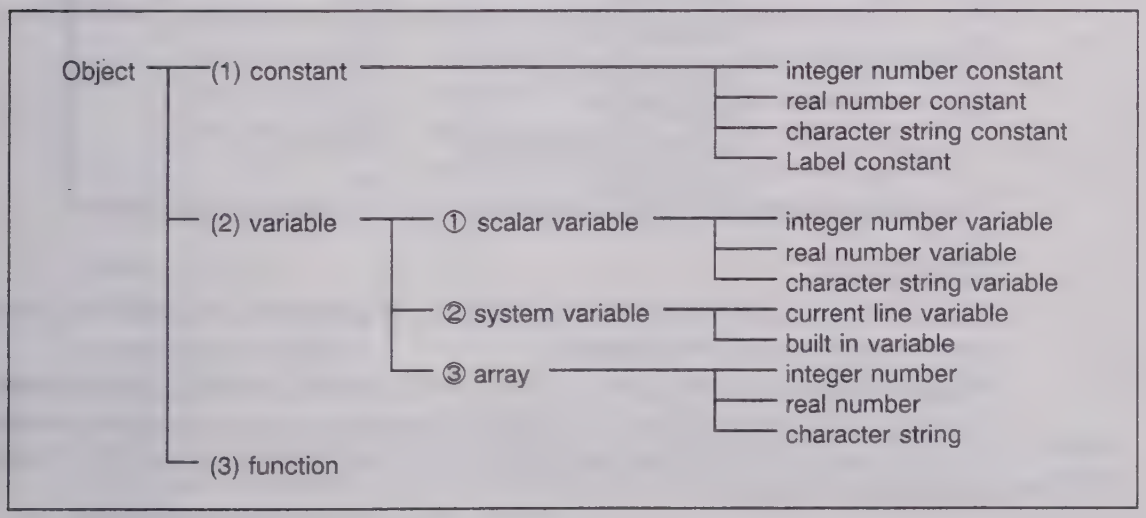
The expression consists of the object and operator and can be placed anywhere it can be grammatically specified to. (However, since the condition expression of IF statement interpret the symbol " = " as equal sign because of the compatibility with the conventional BASIC, the assignment expression cannot be written.)
There are four kinds of expressions, depending on which kinds of data type is used for the final value as a result of computation.

< arithmetic expression > < character string expression > < logical expression > < label >

Arithmetic expression: Results in an integer value or real value,
logical expression: Is determined by the syntax regardless of whether the expression includes the logical operator within itself and estimates the final value as logical value, i.e., "0" is false and "1" is true.

4.1.2 Object

The item to be processed by BASIC is called "object". The object may be a constant, variable, and function and each object type consists of:



(1) Constant

●Integer number constant
The constant which has no decimal point within a program is considered as an integer number. Since the constant is represented using four bytes inside, it can range from -2,147,483,648 to +2,147,483,647.

●Real number constant

The constant which has a decimal point or is represented using a floating decimal point such as 1E+20 is considered as a real number. Since the constant is represented using eight bytes (IEEE) inside, it can range from approx. -1E+308 to approx. 1E+308 and has an accuracy of 15 digits.

●Character string constant

To represent a character string, it must be enclosed with double quotation marks ("). It is possible to specify any character string between the empty string "" and a maximum of 128 character string. The unit of the included character is 8 bits and it is possible to represent up to 256 kinds of character units of 0 to 255. ASCII codes are used as character codes, which register special symbols to codes from 128 to 255.

For the program to represent the codes which are not assigned to the keyboard or to enter the INPUT statement, the form field (\f) method is prepared using "\". Similarly, "\" can be written to include the double quotation mark " into the character string.

To represent the ASCII control characters, escape sequences are prepared, as follows:

Escape sequences	Meanings	total number	Decimal number
\b	Back space	010	8
\t	Horizontal TAB	011	9
\n	Line field (new line)	012	10
\v	Vertical TAB	013	11
\f	Form field (clear screen)	014	12
\r	Carriage return	015	13

●Label constant

Label can be used instead of the line number. For declaration, an asterisk (*) should be added to the beginning of the program.

The usable character is the same as the variable. However, since it is not a variable, any character cannot be substituted. In addition, the positions where the label can be written are limited to the line number part described in "4.3 Statement Syntax and Use" or the part where "label" is written.

(2) Variable

The name of variable consists of up to 20 alphanumeric characters, starting with an alphabetic character.

If the last character of the variable name is $:	Character string variable
If the last character is (integer):	Array type variable If INTEGER statement does not declare the variable type, the variable is used as a real number type.

Table 4-3 Alphanumeric Characters

1, 2, 3, 4, 5, 6, 7, 8, 9, 0
a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, l, r, s, t
u, v, w, x, y, z
A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T
U, V, W, X, Y, Z
-

Example: Variable types

Value, A123 :	Numeric variable (Real type)
Str$, K67$:	Character string variable (18 characters)
INTEGER Val, M00 :	Numeric variable (Integer type)
INTEGER Ary(10) :	Numeric value type array variable (Integer type)
DIM Ss$[100] :	Character string variable (100 characters)
DIM Sarr$(5)[10] :	Character string type array variable (Five times ten characters)
DIM Array(3) :	Numeric value type array variable (Real type)

① Scalar variable

- Integer number variable
- Real number variable
- Character string variable

As long as the variable is not initialize, "0" is assigned to the numeric type variable. Therefore, if the variable is to be initialized to a specific value, it is necessary to specifically substitute a value in the program.
The value which can be stored each data type has the same amplitude as for the constant. The character string has the length attribute similarly to the character string constant. To declare the length, DIM statement should be used.

DIM string$[100]

If the reference is made without the declaration, the variable is considered as 18 character string. A part of the character string can be handled using the sub-string operator ([ ]).

Refer to "(7) Sub-string operator" in section 4.1.3.

```
string$ = "ADVANTEST CORPORATION"
PRINT string$[1,14] ; ","
```

Result

```
ADVANTEST CORP,
```

② System variable

The current line variables (@) and the built-in variables (PI) are available for the system variables. The current line variables (@) are the variables that store the line number of the program that is currently performed. And the built-in variables (PI) return the approximate value of the circular constant 3.14159265359.
And, any value cannot be substituted because these variables are the reserved word.

③ Array

For declaration of the array, use DIM or INTEGER statement. The numeric value type array variables and the character string type array variables are not able to be used if the array is not be declared. Before using, the required area should be declared with INTEGER or DIM statement about the numeric value type array variables and with the DIM statement about the character string type array variables. The error message is displayed when they are used without the declaration.
The number of the dimension of the array that is able to be specified both in the numeric value type array variables and in the character string type variables is up to the five dimensions. In this case, the attached character is always assigned starting at 1.

INTEGER A(3,5)	Numeric value type array variables (Two dimensional array of the real number type)
DIM A(3,4,5,6)	Numeric value type array variables (Four dimensional array of the integer number type)
DIM B$(5)	Character string type array variables (A linear array of the five times of 16 character string)
DIM B$(3,5) [128]	Character string type array variables (Two dimensional array of three times five of 128 character string)

(3) Functions

All the functions are built in type and grouped into the integer number type, real number type, and character string type, depending on its return value. In addition, since the function call can be written in an operation expression, it can be handled similarly to the variable.

```
string$ = "ADVANTEST"
PRINT string$
A = NUM("A")
Z = NUM("Z")
FOR i=1 TO LEN(string$)
    b = NUM(string$(i)) - A + Z
    string$(i)=CHR$ b
NEXT i
PRINT string$
```

Result

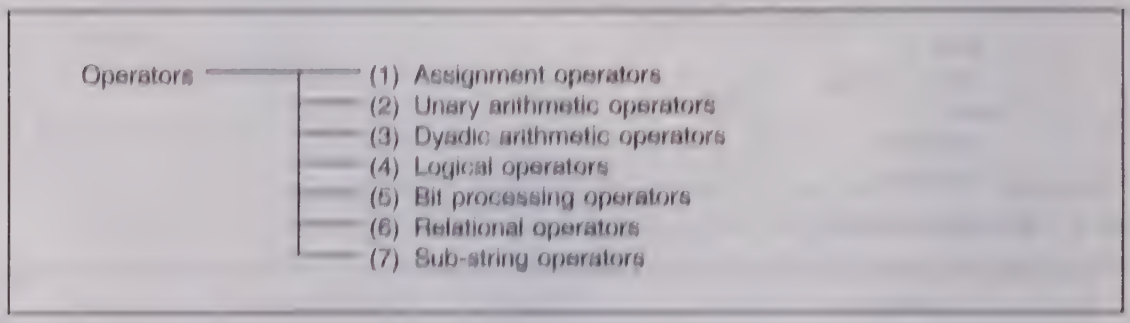
```
ADVANTEST
ZDVZNTEST
```

- Built-in functions

Functions	Descriptions
SIN (Arithmetic expression) COS (Arithmetic expression) TAN (Arithmetic expression) ATN (Arithmetic expression)	Sine (sin) Cosine (cos) Tangent (tan) Reverse tangent (tan^{-1}) Unit of angle = radian
LOG (Arithmetic expression)	Natural logarithm
SQR (Arithmetic expression)	Square root
ABS (Arithmetic expression)	Absolute value
NUM (Character string expression)	Returns ASCII code for the first one character of the character string expression. Example: NUM ("A") —> 65
CHR$ (Arithmetic expression)	Returns the character string of the ASCII code one character corresponding to the value of the arithmetic expression. Example: CHR$ (65) —> "A"
LEN (Character string expression)	Returns the length of the character string expression. Example: LEN ("ADVANTEST") —> 8
POS (Arithmetic expression 1, Arithmetic expression 2)	Returns the digit of the head character of the character string corresponding to the character string expression 2 in the character string expression 1. Example: POS ("ADVANTEST", "AN") —> 4

4.1.3 Operators

Operator are used to operate the object operand. An expression is coded by combining operators and objects.



(1) Assignment operators

The key word existed in the standard BASIC, which is called "LET" is not provided for the assignment operator. Assignment expression contains has its values and and makes up an expression.

```
PRINT a=1              ---> 1.0
PRINT a$="ADVANTEST"   ---> "ADVANTEST"
PRINT (a=1)+a          ---> 2.0
```

The assignment operators are shown below:

=: Normal assignment
In the assignment for character-string variables, transmits the only effective value of right part.
Example: DIM string$ [20]
PRINT LEN (string$ = "12345")
Result
5

=: Converts the value depending on the data type of left part, then assigns it to variable.
Example: string$ = 123.456 ---> "123.456"
numeric = "123" ---> 123
integer = 123.456 ---> 123

+=: a += 10 ---> a = a + 10
-=: a -= 10 ---> a = a - 10
*=: a *= 10 ---> a = a * 10
/=: a /= 10 ---> a = a / 10
%=: a %= 10 ---> a = a % 10
=<: Assigns the character strings left-justify to variables.
=>: Assigns the character strings right-justify to variables.

(2) Unary arithmetic operators

- : Minus sign
+ : Plus sign
++ : Front/Back increment
  Front b = ++a ... Adds 1 to a, then assigns ++a to b.
  Back b = a++ ... Assigns a++ to b, then adds 1 to a.
-- : Front/Back Decrement
  Front b = --a ..... Subtracts 1 from a, then assigns --a to b.
  Back b = a-- ..... Assigns a-- to b, then subtracts 1 from a.

Example: a = 10: PRINT a++: PRINT a: PRINT --a: PRINT --a: print a
Result
10
11
10
9
9

(3) Dyadic arithmetic operators

+: Addition
-: Subtraction
*: Multiplication
/: Division
%: Modulo calculation (remainder)
^: Involution
&: Coupling characters

(4) Logical operators

NOT AND OR XOR

(5) Bit processing operators

In numeric expressions, only the integer type is available. Real type may result in an error.

BNOT BAND BOR BXOR

(6) Relational operators

The following operators are provided, and the result of applying these operators is a boolean value, either TRUE or FALSE. At this case, TRUE is 1, and FALSE is 0. When the relational operation is resulted based on the BASIC syntax, if the value calculated finally resulted in 0, the result is determined as FALSE. All the values other than calculated values become TRUE.

=: Equal
<>: Not equal (or !=)
<
>
<=
>=

Since the relational operations always perform the arithmetic operation according to the IF statement condition, the operator "=" is determined unconditionally as relational operator. Therefore, the assignment expression cannot be included in the IF statement conditional expression.

(7) Sub-string operators

Enables to specify the character-string expression in part as character string.

Character-string expression [arithmetic expression 1, arithmetic expression 2]:
The sub-string operator is considered (defined) as from.
"ADVANTEST" [1,5] ---> "ADVAN"

Character-string expression [arithmetic expression 1, arithmetic expression 2]:
The sub-string operator is considered (defined) as from.
"ADVANTEST" [6;4] ---> "TEST"

- List of the order of precedence of the operators

Operator	Order of Precedence	Classification
( )	1	
= += -= *= /= %= =< =>	2	Assignment operator
OR XOR	3	Logical operator
AND	4	Logical operator
BOR	5	Bit processing operator
BXOR	6	Bit processing operator
BAND	7	Bit processing operator
= == <>	8	Relational operator
< > <= >=	9	Relational operator
- + &	10	Dyadic arithmetic operator
* / %	11	Dyadic arithmetic operator
NOT	12	Logical operator
- +	13	Unary arithmetic operator
^	14	Dyadic arithmetic operator
A++ A--	15	Unary arithmetic operator
++A --A	16	Unary arithmetic operator
BNOT	17	Bit processing operator

- Examples to be noted

```
10 A=1 : B=2
20 IF ((A=1) AND (B=2)) THEN ..... ○
20 IF A=1 AND B=2 THEN ..... ×
```

In the compared examples as mentioned above, the format marked with "x" cannot get the expected results because AND is processed before the equal (=) because of their precedence. The parentheses should always be used to be well-defined the precedence of the process.

4.2 Various Statements

4.2.1 Statement Function List

(1) Basic (fundamental) statement

Statement	Function
ABS	Returns the absolute value.
ATN	Returns the inverse tangent value.
CHR$	Returns the values that have the specified character cord.
CLS	Clears the screen.
COLOR	Specifies the color of the characters on the screen.
COMMON	Delivers the variables of the merged program.
CONSOLE	Specifies the scroll area.
COS	Returns the value of a cosine.
CSRLIN	Returns the position coordinate y.
CSRPCS	Returns the position coordinate x.
CURSOR	Moves the cursor.
DATA	Defines the numeric value or character string to be read out by READ statement.
DATE$	Reads out the date of timer (RTC) built into the analyzer.
DIM	Defines the array variable or character-string variable.
DISABLE INTR	Disables the acceptance of the interruption.
ENABLE INTR	Enables the acceptance of the interruption.
ERRM$	Returns the error message.
ERRN	Returns the error number.
EXP	Obtains a value of the argument power of constant e and returns that.
FOR-TO-SETP, NEXT, BREAK, CONTINUE	Executes the loop processing.
FRE	Returns the BASIC program buffer remaining capacity.
GOSUB, RETURN	Branches or returns to the subroutine.
GOTO	Branches to the specified line.
GPRINT	Outputs to the numeric value or character string to the GPIB.
IF-THEN, ELSEL END IF	Conditional branch
INKEY$	Returns the cord of the panel key.
INPUT	Inputs from the panel key.
INTEGER	Defines the variable as an integer type.
LEN	Returns the number of digits of the character strings.
LOG	Returns the natural logarithm.
LPRINT	Outputs the numeric value or character string to the parallel port.
NUM	Returns the ASCII cord of the specified character.
OFF ERROR	Cancels the branch when detecting the BASIC error.
OFF ISRQ	Cancels the interruption branch by ISRQ.

(Cont'd)

Statement	Function
OFF KEY	Cancels the interruption branch by key input.
OFF SRQ	Cancels the interruption branch by SRQ.
ON DELAY	Branches after the specified time elapses.
ON ERROR	Defines the branch when detecting the BASIC error.
ON ISRQ	Defines the interruption branch by the internal request.
ON KEY	Defines the interruption branch by key input.
ON SRQ	Defines the interruption branch by externally GPIB SRQ.
PRINT [USING]	Displays the numeric value or character string.
PRINTER	Sets the printer GPIB address.
PRINTF	Displays the numeric value or character string.
READ	Assigns the constant of DATA statement to the variable.
REM	Annotation
RESTORE	Specifies the data line to be read in next READ statement.
SELECT, CASE, END SELECT	Executes the multi branches with condition of expression value.
SPRINTF	Assigns the result according to PRINTF format to the character string.
TIME$	Returns the value of timer (RTC) built into the analyzer.
TIMER	Reads out and resets the value of the built-in system timer.
WAIT	Waits for the specified time.
WAIT EVENT	Waits for the occurrence of the specified event.

(2) GPIB control statement

Statement	Function
CLEAR	Clears the device.
DELIMITER	Specifies the block delimiter.
ENTER	Inputs from the GPIB.
INTERFACE CLEAR	Clears the GPIB interface.
LOCAL	Cancels the remote control.
LOCAL LOCKOUT	Local lockout
OUTPUT	Outputs to the GPIB.
REMOTE	Remote control
REQUEST	Sets the status byte.
SEND	Outputs (sends) the command, data, and others to the GPIB.
SPOLL	Reads out the status byte.
TRIGGER	Outputs the group-execute trigger.

(3) File control statement

Statement	Function
CLOSE	Closes the file.
DSTAT	Obtains the directory contents of memory card for the BASIC variable.
ENTER [USING]	Reads out the data from the file.
OFF END	Cancels the processing specified by ON END statement.
ON END	Defines the processing at the end of file.
OPEN	Opens the file.
OUTPUT [USING]	Outputs (writes) the data to the file.

4.2.2 Statement Syntax List

(1) Basic statement

Statement	Syntax
ABS	ABS (numeric expression)
ATN	ATN (numeric expression)
CHR$	CHR$ (numeric expression)
CLS	CLS
COLOR	COLOR <Color of the character-string> [,reverse mode]
COMMON	COMMON numeric variable | character string variable | array variable
CONSOLE	CONSOLE <start line> <last line>
COS	COS (numeric expression)
CSRLIN	CSRLIN
CSRPOS	CSRPOS
CURSOR	CURSOR <X axis> <Y axis>
DATA	DATA numeric constant | character-string constant {, numeric constant | character-string constant}
DATE$	(1) DATE$ (2) DATE$ = "YY/MM/DD"
DIM	DIM <B>|<C> {, <B>|<C>}
DISABLE INTR	DISABLE INTR
ENABLE INTR	ENABLE INTR
ERRM$	ERRM$ (error number)
ERRN	ERRN

B: numeric variable name [(numeric expression {, numeric expression})]

C: character-string variable [numeric expression]

(Cont'd)

Statement	Syntax
FOR-TO-SETP, NEXT, BREAK, CONTINUE	FOR numeric variable = numeric expression TO numeric expression [STEP numeric expression] [BREAK] [CONTINUE] NEXT [numeric variable]
EXP	EXP [(numeric expression)]
FRE	FRE (numeric)
GOSUB, RETURN	GOSUB integer | label expression RETURN
GOTO	GOTO integer | label expression
GPRINT	GPRINT [A {, | ;A}]
IF-THEN, ELSEL END IF	(1) IF<conditional expression> THEN <statement> (2) IF<conditional expression> THEN [ELSE IF <conditional expression> THEN] [multi statements] [ELSE] [multi statements] END IF
INKEY$	INKEY$
INPUT	INPUT ["<character-string>",] A {, A}
INTEGER	INTEGER <B> {, <B>}
LEN	LEN (character-string expression)
LOG	LOG (numeric expression)
LPRINT	LPRINT [A {, | ;A}]
NUM	NUM (character expression)
OFF ERROR	OFF ERROR
OFF ISRQ	OFF ISRQ
OFF KEY	OFF KEY [key code]
OFF SRQ	OFF SRQ
ON DELAY	ON DELAY time GOTO | GOSUB integer | label
ON ERROR	ON ERROR GOTO | GOSUB integer | label
ON ISRQ	ON ISRQ GOTO | GOSUB integer | label
ON KEY	ON KEY key code GOTO | GOSUB integer | label
ON SRQ	ON SRQ GOTO | GOSUB integer | label

A: numeric expression | character-string expression

B: numeric variable name [(numeric expression {, numeric expression})]

(Cont'd)

Statement	Syntax
PRINT [USING]	(1) PRINT [A {, | ;A}] (2) PRINT USING format setup expression ; {, A}
PRINTER	PRINTER numeric expression
PRINTF	PRINTF format expression {, A}
READ	READ input item {, input item}
REM	REM [character string] or ![character string]
RESTORE	RESTORE integer | label expression
SELECT, CASE, END SELECT	SELECT <numeric expression | character-string expression> CASE <numeric expression | character-string expression> multi statements [CASE ELSE] [multi statements] END SELECT
SPRINTF	SPRINTF character-string variable format specification {, A}
TIMER	TIMER (0|1)
TIME$	(1) TIME$ (2) TIME$ = "HH:MM:SS"
WAIT	WAIT time
WAIT EVENT	WAIT EVENT <event number>

A: numeric expression | character-string expression

- In PRINT USING format specification, specify the following image specifications by using a comma among images.
  image specifications
  - D: Specifies the output digits with No. of D. A space is used to fill up the remaining blank in the specified field.
  - Z: Specifies the output digits with No. of Z. A zero is used to fill up the remaining blank in the specified field.
  - K: Displays the expression as it is.
  - S: Displays the PRINT USING format with a + or - sign flag at the position of S.
  - M: Displays the PRINT USING format with a - for negative and a space for positive at the position of M.
  - .: Displays the PRINT USING format to match the position "." with coming the decimal point.
  - E: Displays PRINT USING format with the exponent format (e, sign, exponent).
  - H: Same as K. However, use a comma for a decimal point.
  - R: Same as ".". However, use a comma for a decimal point.
  - *: Specifies the output digits with the number of *. A space is used to fill up the remaining blank in the specified field.
  - A: Displays one character.
  - k: Displays the character-string expression as it is.
  - X: Displays the character of one space.
  - Literal: Encloses a literal with \" when writing it to the format expression.
  - B: Displays the expression result using an ASCII code.
  - @: Form lead
  - +: Moves the display position to the top of the same line.
  - -: Line feed
  - #: Does not line feed.
  - n: Specifies the number of repetition of each image by using numerics.

- In PRINTF format specification, specify the parameter immediately followed after % by using the following image.

  %[-] [0] [m] [. n] character

  -: Justifies the character with no space from left (if no specification, then from right).

  0: Sets the character, which is justified for the remaining blank in the specified field, to be 0.

  m: Reserves the field for the character "m".

  .n: Outputs the PRINT USING format with n-digit accuracy. In character string, this setup value is used for an actual character-string length.

  Character: d; decimal with sign s; character string
  o; octal e; floating-point expression (exponent format)
  x; hexadecimal f; floating-point expression

(2) GPIB statement

Statement	Syntax
CLEAR	CLEAR [unit address {, unit address}]
DELIMITER	DELIMITER numeric expression
ENTER	ENTER unit address ; B {, B}
INTERFACE CLEAR	INTERFACE CLEAR
LOCAL	LOCAL [unit address {, unit address}]
LOCAL LOCKOUT	LOCAL LOCKOUT
OUTPUT	OUTPUT unit address {, unit address} ;A {, A}
REMOTE	REMOTE [unit address {, unit address}]
REQUEST	REQUEST integer
SEND	SEND CMD | LISTEN | TALK [numeric expression, ···]
	SEND DATA [numeric expression | character expression | EOI]
	SEND UNL | UNT
SPOLL	SPOLL (unit address)
TRIGGER	TRIGGER [unit address {, unit address}]

A: numeric expression | character-string expression

B: numeric variable [character-string expression

(3) File control statement

Statement	Syntax
CLOSE	CLOSE #FD | *
DSTAT	(1) DSTAT 0 <number of file> (2) DSTAT <index> <file name> <attribute> <size> <number of sector> <year> <month> <date> <time> <minute> <start sector> (3) DSTAT ;SELECT <character string> COUNT <variable>
ENTER [USING]	(1) ENTER #FD ; input item {, input item} (2) ENTER #FD USING "image specification" ; input item {, input item} }
OFF END	OFF END #FD
ON END	ON END #FD GOTO|GOSUB integer|label expression
OPEN	OPEN "file name" FOR processing mode AS #FD [; type]
OUTPUT [USING]	(1) OUTPUT #FD ; output item {, output item} (2) OUTPUT #FD USING "image specification" ; output item {, output item} }

FD: file descriptor
Processing mode: INPUT|OUTPUT
Type: BINARY|TEXT|ASCII

- ENTER USING image specification
  image specification
  - D: Interprets the numeric of D as an input digit and reads out it, then assigns it to the variable of the input item.
  - Z: Same as D.
  - K: Reads one line and converts it to the numeric data, then assigns it to the variable of the input item.
  - S: Same as D.
  - M: Same as D.
  - .: Same as D.
  - E: Same as K
  - H: Same as K. However, use a comma for a decimal point.
  - *: Same as D.
  - A: Reads the number of A and assigns it to the character-string variable.
  - k: Reads one line and assigns it to the character-string variable.
  - X: Skips one character.
  - Literal: Skips the the character-string numeric data enclosed with \".
  - B: Reads one character and assigns it to the input item using an ASCII code.

@: Skips one-byte data.
+: Same as @.
-: Same as @.
#: Ignored in ENTER statement.
n: Specifies the number of repetition of each image by using numerics.

- OUTPUT USING image specification

image specification

D: Specifies the output digits with No. of D. A space is used to fill up the remaining blank in the specified field.
Z: Specifies the output digits with No. of Z. A zero is used to fill up the remaining blank in the specified field.
K: Displays the expression as it is.
S: Displays the OUTPUT USING with a + or - sign flag at the position of S.
M: Displays the OUTPUT USING with a - for negative and a space for positive at the position of M.
.: Displays the OUTPUT USING to match the position "." with coming the decimal point.
E: Displays OUTPUT USING with the exponent format (e, sign, exponent).
H: Same as K. However, use a comma for a decimal point.
R: Same as ".". However, use a comma for a decimal point.
*: Specifies the output digit with the number of *. A space is used to fill up the remaining blank in the specified field.
A: Displays one character.
k: Displays the character-string expression as it is.
X: Displays the character of one space.
Literal: Encloses the literal with \" when writing it in the format expression.
B: Displays the expression result using an ASCII code.
@: Outputs the form lead.
+: Outputs the carriage return.
-: Outputs the line feed.
#: Does not hang the line feed immediately followed after the last item.
n: Specifies the number of repetition of each image by using numerics.

4.3 Statement Syntax and Use

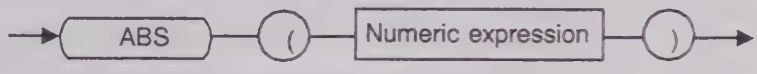
1. ABS

Outline

The ABS is a built-in function that is used to return the absolute value.

Syntax

(1)-1



(1)-2

ABS (Numeric expression) →

Example

```
PRINT ABS(-2)
A=ABS(B)
```

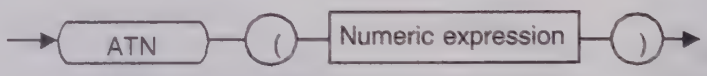
2. ATN

Outline

The ATN is a built-in function that is used to return the inverse tangent (arctangent) value.

Syntax

(1)-1



(1)-2

ATN (Numeric expression) →

Example

```
PRINT ATN(PI/2)
A=ATN(-3.14/B)
```

Note : The ATN statement returns the values that are in the $-\pi/2$ to $\pi/2$ range in radian.

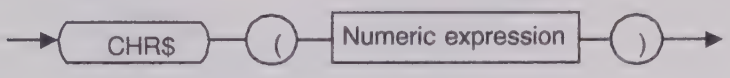
3. CHR$

Outline

The CHR$ is a built-in function to return the value that have the specified character code.

Syntax

(1)-1



(1)-2

CHR$ (Numeric expression) →

Example

```
PRINT CHR$(65)
A$=CHR$(B)
```

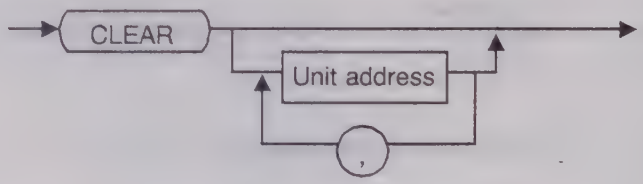
4. CLEAR

Outline

The CLEAR statement is used to set the all units connected to a GPIB or the selected particular units to an initial state. In other word, this statement clears the all setup values for units.

Syntax

(1)-1



(1)-2

CLEAR [unit address {, unit address}]

Description

- If only the CLEAR statement is performed without specifying the unit address, the universal Device Clear (DCL) command will be sent. By the DCL command, all the units, which is connected to a GPIB, could be set to the initial state.
- When the unit address is specified followed after the CLEAR statement, only the units which are specified by the unit address are addressed, then the Select Device Clear (SDC) command is sent. By the SDC command, only the particular units is set to the initial state. Multiple unit-address can be specified.
- The initial state that is defined for each unit in the CLEAR statement depends on each unit.

Example

```
10 CLEAR
20 CLEAR 2
30 CLEAR 1, 3, 5, 7
```

Note

The CLEAR statement is not available in ADDRESSABLE mode.

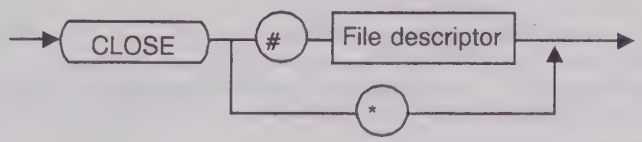
5. CLOSE

Outline

The CLOSE statement is used to close files assigned to a file descriptor.

Syntax

(1)-1



(1)-2

CLOSE <#file descriptor| *>

Description

- All files opened by the OPEN command must be closed before removing a memory card or turning off the power of units. If not, the files may be damaged.
- In BASIC program, when operation is suspended using the PAUSE or STOP key, files are not closed automatically. In other cases, all files are closed automatically after programming, also after termination with an error. However, if ON ERROR is set in instrument, the files will not be closed. By reasons above, be sure to perform the close operation certainly by using the following method (specification method for closing all files using the command) at the error termination.

  CLOSE *

- The files are closed automatically when command such as SCRATCH or LOAD is executed.

Note: For the information how to handle files, refer to "2.4 File Management".

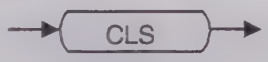
6. CLS

Outline

The CLS statement is used to clear the display on the screen.

Syntax

(1)-1



(1)-2

CLS

Description

- The CLS statement clears the characters displayed on the screen and immediately returns the cursor to the original position.
- The CLS statement clears the scroll range specified by CONSOLE.

Example

```
10 CLS
```

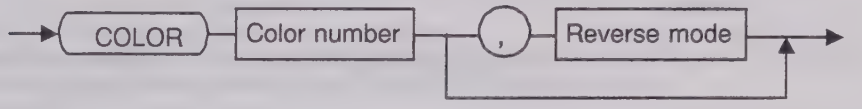
7. COLOR

Outline

The COLOR statement is used to specify the color of characters output on the screen (only when the BASIC screen is selected). The color or the reverse mode specified by this command continue till they are reset.

Syntax

(1)-1



(1)-2

COLOR <Color of characters> [,Reverse mode]

Available Range

Color of characters: [0] Black [1] Red [2] Green [3] Yellow
[4] Blue [5] Purple [6] Light blue [7] White

Reverse mode: [0] Normal video [1] Reverse video

Description

COLOR 3 → Specifies the color of characters in yellow.

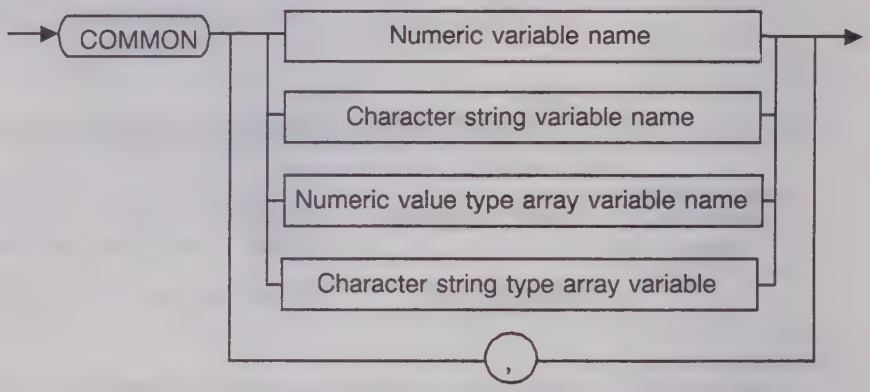
8. COMMON

Outline

The COMMON statement is used to deliver the variables from the program on the memory to the program merged to the program on the memory by the MERGE command. The COMMON statement is usable after the declared program line.

Syntax

(1)-1



(1)-2

COMMON Numeric variable | Character string variable | Numeric value type array variable | Character string type array variable[, ...]

Description

COMMON A, B$, C(), D$() → The variables of the numeric variable A, the character variable B$, the numeric value type array variable C() and the character string type array variable D$() are delivered to the merged program.

Note

- It is impossible to refer the value of the local variable used in the merged program from the command line after execution of the merged program.
- When the COMMON declaration of the numeric value type array variable and of the character string array variable is performed, the declaration of the array should be done before doing it with the DIM or INTEGER statement. If it is performed without the declaration, the error message is displayed.
- The COMMON declaration is able to be performed even on the side of the merged program but the variables to be COMMON declared should have already been declared (performed) on the side of the program existing on the memory. Be careful that the variables that have the same variable names appeared before and after the COMMON declaration are handled as the different variables.

Examples: The examples of the merged programs

```
1000 A=100 ....  Local variable
1010 PRINT A ➔   100.0
1020 COMMON A  .  Common variable (Handled as the different variable from
                  the variable A on the line 1000.)
1030 PRINT A ➔   0.0
```

* The variables have the different meanings before and after the COMMON declaration even if they have the same variable name A.

- When the COMMON declaration is performed on the merged program, the array should be declared by the DIM or INTEGER statement on the side of the program on the memory in advance. The error message is displayed if it is handled without the declaration.

```
SELECT Value
CASE 1                  → When the numeric variable Value is 1.
  PRINT ''TYPE-1''
  Mode=100
CASE 2,3                → When the numeric variable Value is 2 or 3.
  PRINT ''TYPE-2''
  Mode=200
CASE A*2+1              → When the numeric variable Value is (the variable A *2 + 1).
  PRINT ''TYPE-4''
  Mode=300
CASE ELSE               → When the numeric variable Value is other than
  PRINT ''TYPE-OTHER''    the type of CASE statement above-mentioned.
  Mode=400
END SELECT
```

9. CONSOLE

Outline

The CONSOLE statement is used to specify the scroll range.

Syntax

(1)-1

(1)-2

CONSOLE <start line, output number of lines>

Available Range

Start line: 0 to 26
Output number of lines: 1 to 27

Description

CONSOLE 0, 27 →The output range is specified as the top line 0 to the 27th line.
CONSOLE 5, 10 →The output range is specified as the 6th line to the 10th line.

Note

- The range of screen to be output specified with the CONSOLE command is usable only about the now-active screen. That is to say, there are the different information of the output range for the waveform screen and for the BASIC screen.
- The output will not be done beyond the output range specified by the CONSOLE statement basically. When beyond the output range is specified by the CURSOR, it becomes possible to output beyond the range just after it for once by the PRINT command. And after it outputs once, it returns to the output within the range.

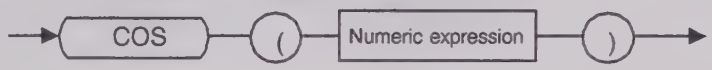
10. COS

Outline

The COS is a built-in function that is used to return the value of a cosine. The unit of the value specified in the numeric expression is specified in radian. (Refer "4.3-2 ATN" and "68. SIN".)

Syntax

(1)-1



(1)-2

COS (numeric expression) →

Example

```
PRINT COS(PI/2)
A$=COS(B)
```

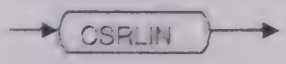
11. CSRLIN

Outline

The CSRLIN statement returns the position coordinate y of the current cursor-position.

Syntax

(1)-1



(1)-2

CSRLIN →

Available Range

Position coordinate y: 0 to 26

Description

`PRINT CSRLIN` → Displays the position coordinate y of the current cursor-position onto the screen.

`Ypos=CSRLIN` → Assigns the position coordinate y of the current cursor-position to the numeric variable Ypos.

12. CSRPOS

Outline

The CSRPOS statement is used to return the position coordinate x of the current cursor-position.

Syntax

(1)-1

(1)-2

CSRPOS→

Available Range

Position coordinate x: 0 to 66

Description

`PRINT CSRPOS` → Displays the position coordinate x of the current cursor-position onto the screen.

`Xpos=CSRPOS` → Assigns the position coordinate x of the current cursor-position to the numeric variable Xpos.

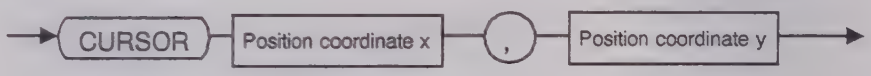
13. CURSOR

Outline

The CURSOR statement is used to move the cursor (the output position) to the coordinate position specified on the screen.

Syntax

(1)-1



(1)-2

CURSOR <Position coordinate x , Position coordinate y>

Available Range

Position coordinate x : 0 to 66
Position coordinate y : 0 to 26

Description

CURSOR 0,0 → Moves the cursor to the coordinate of the upper left of the screen.
CURSOR 5,10 → Moves the cursor to the 6th character on the x coordinate and 11th line on the y coordinate on the screen.

Note

When both positions on the x coordinate and the y coordinate are specified beyond the specified range, it is corrected to be within the specified range.

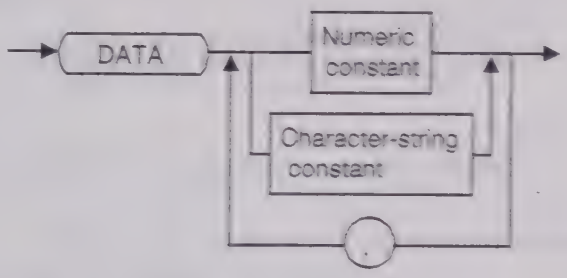
14. DATA

Outline

The DATA statement is used to define the numeric and the character string to be read out by the READ statement.

Syntax

(1)-1



(1)-2

DATA <numeric constant|character-string constant> {, <numeric constant|character-string constant> }

Description

- Since the DATA statement does not become the object to be executed, so it can be placed in any statement number. Generally, the DATA statement is necessary based on the order read out by the READ statement.
- The READ statement searches the DATA statement in the program and retrieves the data to be read.
- To change this order, use the RESTORE statement.
- In DATA statement, multiple constants can be defined, by using commas or spaces for separating the constants. The character string is enclosed with double quotation as character-string constant.
- After the DATA statement, multi-statement separated by a colon cannot be used.

Note

In DATA statement, the parameters (expressions) which include variables cannot be used.

15. DATE$

Outline

The DATE$ statement is used to read out date and to change the date.

Syntax

(1)-1

(1)-2

DATE$ = "

(2)-1

(2)-2

DATE$ = "year/month/day"

Description

- The DATE$ statement reads out the date of the system built-in timer (RTC).
- The read out date can be changed.
  Input as follows:

  DATE$=" 1995/1/1"
  or
  DATE$=" 1995/01/01"

Example

```
10 DIM D$[10]
20 D$=DATE$
30 PRINT "Date is ":D$
40 PRINT "Date Reset"
50 DATE$="1995/1/1"
60 STOP
```

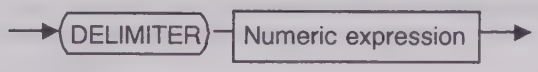
16. DELIMITER

Outline

The DELIMITER statement is used to select four types of delimiters and to set them.

Syntax

(1)-1



(1)-2

DELIMITER numeric expression

Description

- The DELIMITER statement sets the delimiter corresponding to the number resulted by numeric expression.
  The following table shows the selection numbers and the types of delimiters.

Selection No.	Type of delimiter
0	Outputs 2-byte code of CR and LF. Also outputs single signal EOI immediately with LF output.
1	Outputs 1-byte code of LF.
2	Outputs single signal EOI immediately with end of data byte.
3	Outputs 2-byte code of CR and LF.

- If the result of numeric expression exceeds the range of 0 to 3, an error may occur.
  Numeric digits that follow after a decimal point are ignored and recognized as an integer.
- "DELIMITER = 0" is automatically set as a default value when the power is turned on.

Example

```
10 DELIMITER 0
20 DELIMITER 1
30 DELIMITER A*10
```

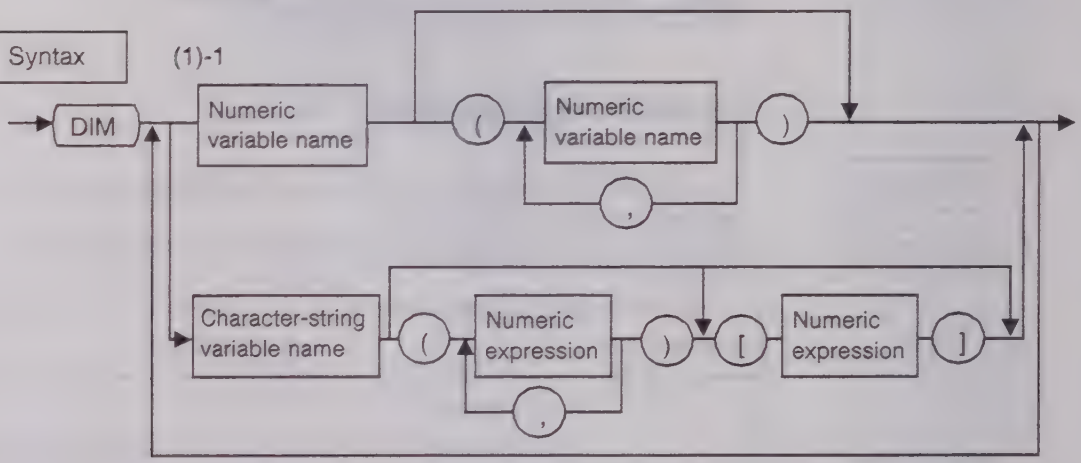
17. DIM

Outline

The DIM statement is used to define the size of the factor of the array variables and assign to the area of the memory. It is able to be declared that the numeric value type array variable to the 10th dimensions in the real number type. And it is possible to be defined that the character string array variable to the 10th dimensions in the same way as the numeric value type array variable.

Syntax

(1)-1



(1)-2

DIM <numeric variable name (numeric expression [, numeric expression, ...]) >
{, numeric variable name (numeric expression [, numeric expression]) }

DIM <character string variable name (numeric expression [, numeric expression, ...])
numeric expression > {, character string variable name (numeric expression
[, numeric expression, ...]) [[numeric expression]] }

DIM <character string variable name [numeric expression] >
{, character string variable name [numeric expression] }

Description

DIM Buf(100) → Gets the area for the 100 real numbers on the memory.
DIM Buf(5,3) → Gets the area for the five times three real numbers on the memory.
DIM Str$(5) → Gets the area for the five times 18 characters on the memory.
DIM Str$(5,3)[10] → Gets the area for the five times three 10 characters on the memory.
DIM Str$[128] → Gets the area for the 128 characters on the memory.

Note

- When the array variable or the character string variable is used, the size of the factor of the array variable should be defined by the DIM statement in advance. However the numeric value type array variable is used with no declaration, it gets the area for the 10 prime numbers in one dimension (same as DIM A(10)), and the character string variable gets the are for the length of the 18 characters (same as DIM A$[18]).
  And the character string array variable is not able to be used without the declaration.
- The numeric expression that indicates an array variable size recognizes the real number as an integer by omitting the digit followed after a decimal point, even if the real number type variable is specified.

Example

```
10 DIM N(5)
20 FOR I=1 TO 5
30      N(I)=I*I/2
40 NEXT I
50 FOR I=1 TO 5
60      PRINT N(I)
70 NEXT
```

18. DISABLE INTR

Outline

The DISABLE INTR statement is used to prohibit the interruption reception.

Syntax

(1)-1

(1)-2

DISABLE INTR

Description

- The DISABLE INTR statement prohibits the interruption by ENABLE INTR statement.
- When the interruption is permitted again after the DISABLE INTR statement performs, the ENABLE INTR statement must be performed. At this case, the branch condition set by ON XXX statement is kept as the previous condition. However, if the condition of interruption branch is changed, it can be set using ON XX or OFF XXX statement before the ENABLE INTR performs.
- After immediately executing (running) the program, the interruption is prohibited until the ENABLE INTR is executed.

Example

```
10 ON KEY 1 GOTO 60
20 ENABLE INTR
30 ! LOOP
40 GOTO 30
50 !
60 DISABLE INTR
70 PRINT "KEY 1 INTERRUPT"
80 STOP
```

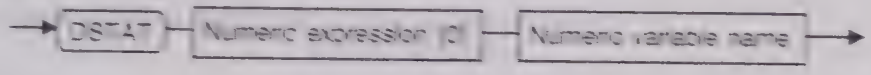
19. DSTAT

Outline

The DSTAT statement is used to obtain the data of the current directory for the BASIC variable. Use the CHDIR command to change the current directory (the current drive).

Syntax

(1)-1

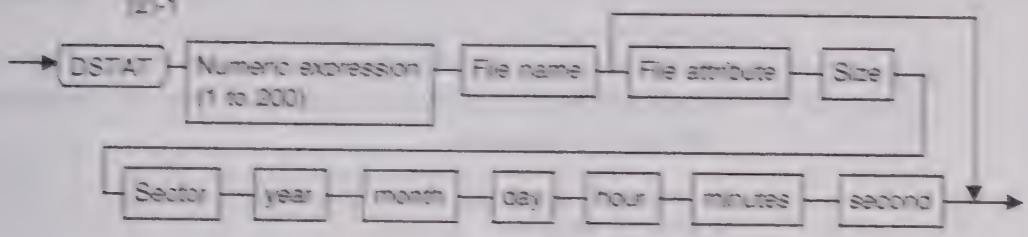


(1)-2

DSTAT <index 0> variable

When 0 is specified for the index, the number of the files stored in the current directory is checked and the number is assigned to the variable.

(2)-1

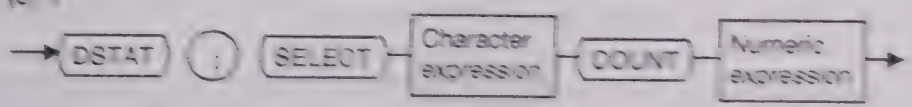


(2)-2

DSTAT <index 1 to 200><file name>
[, file attribute][, file size][, sector'][, file year'][, month'][, day'][, hour][, minutes][, second]

If the value set as the index is within 1 to 200, each file information stored in the current directory is assigned to each variable. The index number here means the registered number registered in the directory.
(It is in the same order displayed by the CAT command.)

(3)-1



(3)-2

DSTAT ; SELECT <character-string> COUNT <variable>

The DSTAT statement assigns the number of the files matching the conditions specified by the parameter <character string> to the parameter <variable>.

The special character expressions that are able to be specified in the parameter <character string> are shown below.

?: Same as one character
*: Same as one character or more
[ ]: Same as any one character in the character string enclosed with [ ]. If the parameter is specified with [A - D], then it is the same as one of the characters of A, B, C or D.

Note: The drive name is not be able to be specified in the parameter <character string>.

Description

DSTAT 0 File_max → Assigns the number of the file in the directory to the variable File_max.

DSTAT 3 File$, Atrb, Size → Assigns the file name with the 3rd registered number in the directory to the variable File$, the file attribute to the variable Atrb and the size to the variable Size.

Description

The file information is loaded for the number of the files registered in the media and it is displayed.

```
10 INTEGER I, Atr,Size,Sector,Year,Month,Day
20 DSTAT 0 Max_file
30 FOR I=1 RO Max_file
40   DSTAT I Name$,Atr,Size,Sector,Year,Month,Day
50   PRINT USING ''k,2D,k,15A,X,2D,X,6*,X,6*,X,4D,k,2Z,k,2Z'';''['',I,'']
                      '',As$,Atr,Size,Sector,Year,''/'',Month,''/'',Day
60 NEXT I
70 END
```

```
<Execution results>
[ 1] AAA.BAS       32 **1598 *****3 1995/03/02
[ 2] BBB.BAS       32 **6326 ****12 1994/11/25
[ 3] ADVAN         48 **    0 *****1 1995/01/06
     ......                      ......
```

File attribute	1. READ ONLY 8. VOLUME LABEL 2. HIDDEN FILE 16. DIRECTORY 3. SYSTEM FILE 32. ARCHIVE FILE
File size	Shows the file size in the number of byte.
Sectors	Number of sector
Year, Month, Day	Date of file created
Hour, minutes	Time of file created

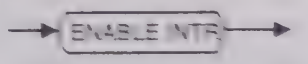
20. ENABLE INTR

Outline

The ENABLE INTR statement is used to permit the interruption reception.

Syntax

(1)-1



(1)-2

ENABLE INTR

Description

- The ENABLE INTR statement permits the interruption reception, and enables the interruption branch defined by ON XXX statement.
- If the interruption is permitted again after performing the DISABLE INTR, then the ENABLE INTER statement must be executed.
- After immediately executing the program, the interruption cannot be performed until the ENABLE INTR statement is performed.

Example

```
10 ON KEY 1 GOTO 60
20 ENABLE INTR
30 ! LOOP
40 GOTO 30
50 !
60 PRINT "KEY 1"
70 GOTO 20
```

CAUTION

If the interruption defined by ON XXX statement occurs, then the interruption cannot be used after immediately the program branches, even if the ENABLE INTR statement is executed (same as DISABLE INTR statement). That is to prevent the nest for the interruption processing, if the next interruption occurred during interruption.
To enable the interruption branch continuously, the ENABLE INTR statement is required again to permit the interruption.

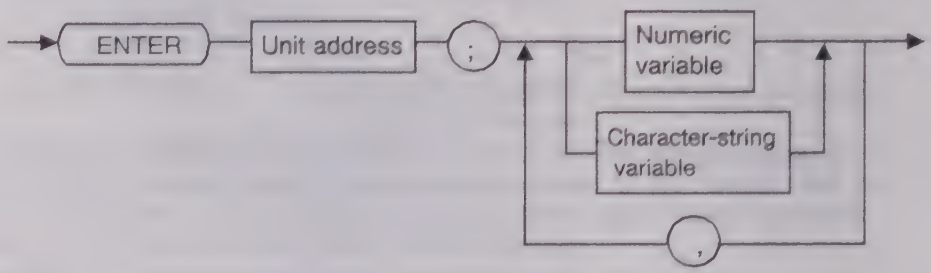
21. ENTER

Outline

(1) The ENTER statement obtains data from a GPIB.
(2) The ENTER statement read data from file and assigns the data to an input item.

Syntax

(1)-1

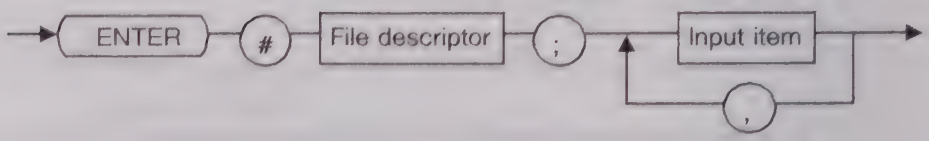


(1)-2

ENTER unit address; <numeric variable|character-string variable>
{, <numeric variable|character-string variable>}

Note: Unit address: 0 to 30; Unit address connected to an external GPIB.
31; Data input from measurement section of the analyzer.

(2)-1



(2)-2

ENTER # file descriptor ; input item {, input item}

Description

Syntax of (1)

- The ENTR statement inputs data from the unit specified by unit address through a GPIB and stores the data into BASIC variable as numeric variable or character string. Pay attention that the controller will stop the operation without completing handshake if talker function is not provided for the unit specified by the unit address.
  When character-string variable is used, it must be defined by DIM statement.
- In character string input, pay attention that the input data will overflow and the overflowed data will be ignored, if the length of character string variable used for destination is not enough.

- Example

```
10 ENTER 1:A
20 DIM A$ (100), B$(20)
30 ENTER 2:A$
40 ENTER 3:B$
```

- Note
  When SYSTEM CONTROLLER mode is selected, the unit specified by the address is set as talker and the data are obtained.

Syntax of (2)

- The ENTER statement reads data as data-type format corresponding input item from the file assigned to the file descriptor, and assigns the data to the input item.

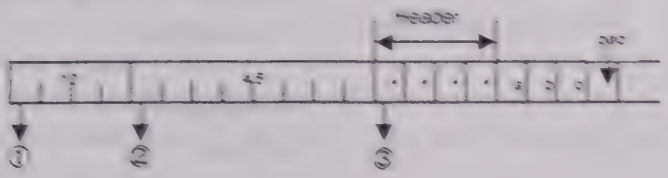
Note: For the information how to handle files, refer to "2.4 File Management".

- Example 1: BINARY file

  The ENTER statement assigns an internal data as it is. It also enables to read the data of the number of byte indicated by the header contents after reading each header such as integer of 4 byte, real number of 8 byte, and character string of 4 byte.
  Since the number of byte to be read is decided by the type of input item, the same type as OUTPUT is required for preventing the data difference

```
10 INTEGER I
20 DIM R
30 OPEN "FILE" FOR INPUT AS #FD
40 ENTER #FD:I,R,S$
```

  Number of byte to be read differs according to the variable type to be assigned.

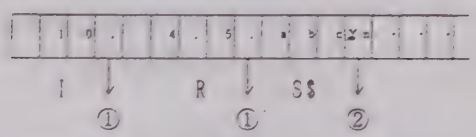


①: When the variable is an integer, 4-byte data is read and assigned to the variable.
②: When the variable is a real number, 8-byte data is read and assigned to the variable.
③: When the variable is a character-string, the 4-byte header and the same-length data as specified by the header are read out. Then the data is assigned to the character-string variable.

- Example 2: TEXT file

Regardless of the number of input items, the TEXT file is read out until the line field. The TEXT file is recognized as one data until a comma and converted into the input-item type, then it is assigned. If the number of input items is more, it cannot be assigned to the variables. Therefore, these values stored in advance are remaining. In reverse, if the number of variables is less than the number of actual data, the data are omitted.

```
10 INTEGER I
20 DIM R
30 OPEN "FILE" FOR INPUT AS #FD;TEXT
40 ENTER #FD;I,R,S$
```

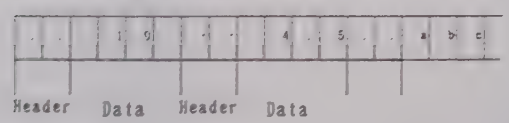


①: Each item is delimited with a string of commas.
②: LF followed after the final item is used.

- Example 3: ASCII file

The 2-byte header and its data according to the header length are read out. The ASCII file is converted into the variable type and assigned.

```
10 INTEGER I
20 DIM R
30 OPEN "FILE" FOR INPUT #FD;ASCII
40 ENTER #FD;I,R,S$
```



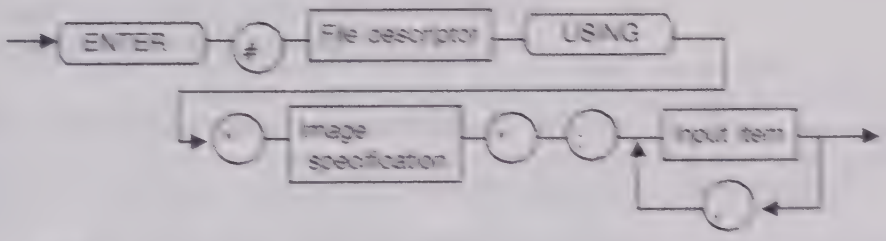
22. ENTER USING

Outline

The ENTER USING statement is used to enter data to the input item from the file by using the image specification format.

Syntax

(1)-1



(1)-2

ENTER # file descriptor USING ' image specification' : input item {, input item}

Note: ENT can be used instead of the ENTER, and USE for the USING.

Description

The ENTER USING statement enters the data to the input item from the file assigned to the file descriptor by using the image specification format.

image specification

D: Recognizes the numeric of D as a numeric digit and reads out it, then assigns it to the variable of the input item.

Z: Same as D.

K: Reads out one line and converts it into the numeric data, then assigns it to the variable of the input item.

S: Same as D.

M: Same as D.

.: Same as D.

E: Same as K.

,: Same as K. However, use a comma for a decimal point.

*: Same as D.

A: Reads the number of A and assigns it to the character-string variable.

X: Reads one line and assigns it to the character-string variable.

X: Skips one-character data.

Literal: Skips the the character-string numeric data enclosed with /".

B: Reads one character and assigns it to the input item using an ASCII code.

image specification

@: Skips one-byte data.

+: Same as @

-: Same as @

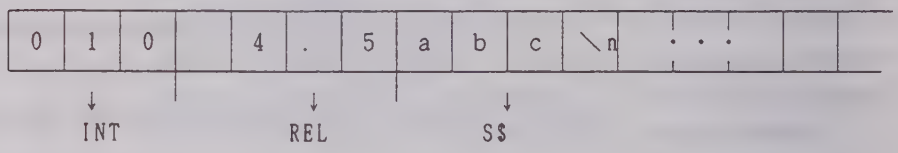
#: Ignored in ENTER statement.

n: Specifies the number of repetition of each image by using numerics. For example, 3D.2D is the same as for DDD.DD, and 4A for AAAA.

Note: For the information how to handle files, refer to "2.4 File Management".

Example

```
10 INTEGER INT
20 DIM     REL
30 ENTER #FD USING "ZZZ,DD.D,3A";INT,REL,S$
```

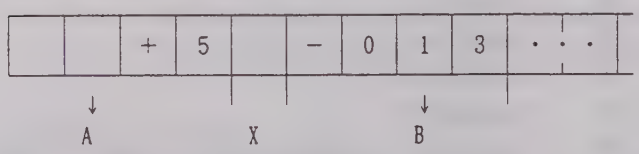


INT: Reads out 3-byte data and converts it into an integer-type data, then assigns it to the variable INT.

REL: The DD.D of image specification corresponds to the REL of the input item. Reads out 4-byte data and converts it into a real-type data, then assigns it to the variable REL. After the execution, the REL becomes 4.5.

S$: Reads out 3-byte data and assigns it to the variable S$. After the execution, the A$ becomes "abc".

```
10 DIM A,B
20 ENTER #FD USING "SDDD,X,MZZZ";A,B
```



A,B: Reads out 4-byte data and converts it into a real-type data, then assigns it to the variables A and B.
After the execution, the A = 5.0, and the B = -13.0.
The image specification X can read 1-byte data, however, cannot assign it to the variable. Converts the data, which is input using an SDDD format, into a real-type data, and assigns it to the variable A.
The image specification X is not required for variable, it skips one character.
The MZZZZ corresponds to the variable B and enters 4-byte data to convert it into a real-type data, then assigns it to the variable B.

```
10 DIM A
20 ENTER #FD USING "K";A
```

S	T	R	I	N	G	1	2	3	.	5	#	#	\n	· ·

Execution result A = 123.5

The STRING123.5## is read out and converted into the real-type data of input variable A. When the input item is a real-type data, the preceding character strings other than numerics, signs (+ , -), and exponents (E, e) are ignored and only the numerics are obtained. Only the numerics can be detected. If the character other than numerics is detected, the conversion is terminated.

For the image specifications such as K, E, k, and H, since LF represents terminator, the data from the current file pointer to the LF as one data are assigned to the variables.

23. ERRM$

Outline

The ERRM$ statement is the system function which is used to return an error message of the number specified.

Syntax

(1)-1

(1)-2

ERRM$ (error number)

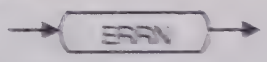
Description

- The ERRM$ statement returns the error message specified by parameters. Particularly, if 0 as a parameter is specified, the ERRM$ returns the error message immediately displayed.
- The error numbers are constructed from as follows:
  Error classes * 256 + error message number
  Error classes: 1; Data input
  2; Data calculation processing
  3; Built-in function
  4; BASIC syntax
  5; Others
- If the numbers which include the error classes are specified, only the error message numbers will be displayed. Therefore, the ERRN can be specified for the error numbers.

24. ERRN

Outline The ERRN statement is the system variable which holds an error number.

Syntax (1)-1



(1)-2

ERRN

Description

- The ERRN statement is the system variable, which holds the error number occurred when the BASIC program is being executed.
- The ERRN is initialized to 0 when the BASIC program starts, and if an error occurs, its number will be assigned to the ERRN. To initialize the assigned value to 0, forcibly assign 0 to the ERRN or re-start the BASIC program.
- The error numbers are constructed from as follows:
  Error classes * 256 + error message number
  Error classes: 1. Data input
  2. Data calculation processing
  3. Built-in function
  4. BASIC syntax
  5. Others

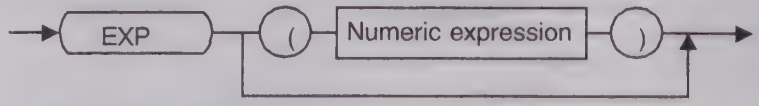
25. EXP

Outline

The EXP is a built-in function that is used to obtain the value of the argument power of constant e (the base of the natural logarithm) and to return that value.

Syntax

(1)-1



(1)-2

EXP[(numeric expression)] →

Description

```
PRINT EXP
PRINT EXP(2)
A=EXP(B)
```

→ Returns the same value as the obtained with the syntax EXP (1).

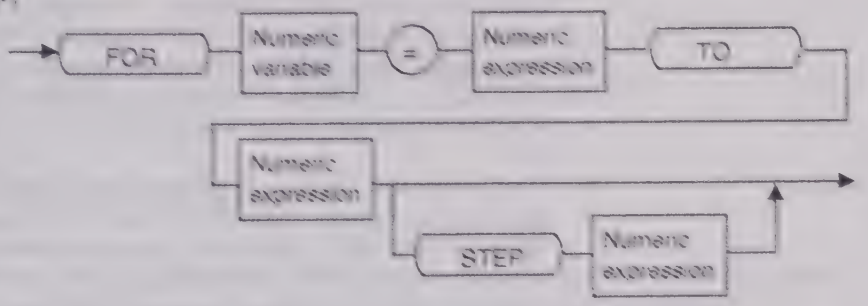
26. FOR - TO - STEP, NEXT, BREAK, CONTINUE

Outline

This statement consists of the program loop (loop processing) by combining with FOR statement and NEXT statement.

Syntax

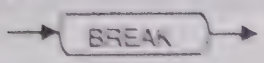
(1)-1



(1)-2

FOR numeric variable = numeric expression TO numeric expression
[STEP numeric expression]

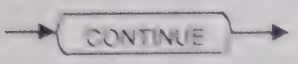
(2)-1



(2)-2

BREAK

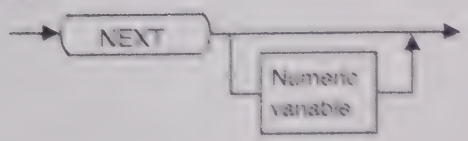
(3)-1



(3)-2

CONTINUE

(4)-1



(4)-2

NEXT [numeric variable]

Description

- This statement uses the numeric variable specified as a loop counter (repetition) and enables to increase the value from the initial value to the final value by the increased step. If the counter value exceeds the final value, then the loop will terminate. The counter increment/decrement is performed by the NEXT statement. Therefore, the program created between FOR statement and NEXT statement is looped repeatedly.
- The values of the initial, final, step are as follows:
  FOR A = (initial value) TO (final value) STEP (increment)
- If STEP (increment) value is omitted, the value is automatically incremented by 1.
- Nest is available between FOR statement and NEXT statement.
- The numeric variable name of the loop counter used for a pair of FOR statement and NEXT statement, be sure to use the same name. If the numeric variable name is different, an error may occur.
- If the value of numeric variable used for the loop counter is changed when the loop processing is executed between FOR statement and NEXT statement, the normal loop processing could not be performed.
- If the numeric variable followed after NEXT statement is omitted, the NEXT statement will automatically correspond to immediately FOR statement.
- BREAK statement can be used to exit in FOR-NEXT loop.
- CONTINUE statement branches to the next step loop in FOR-NEXT loop.

Example

```
10 FOR R=11 TO 0 STEP -5
20   FOR I=0 TO PI STEP PI/180
30     X=SIN(I)*R+23
40     Y=COS(I)*R+15
50     CURSOR X,Y:PRINT "*"
60   NEXT I
70 NEXT R
80 STOP
```

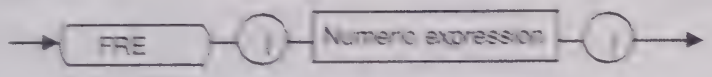
27. FRE

Outline

The FRE statement is the system function which returns the memory space of BASIC buffer.

Syntax

(1)-1



(1)-2

FRE (numeric expression)

Description

- When 0 is specified in the numeric expression, it returns the memory space roughly with the byte number to be used by the BASIC. And when 1 is specified, it returns the memory space with the byte number to be used by the built-in functions.
- This statement checks the memory space roughly and performs no restructure strictly. Therefore, saving and re-loading the data may result in more memory capacity.

Example

```
PRINT FRE(0)
```

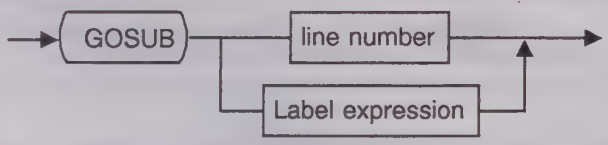
28. GOSUB, RETURN

Outline

This statement is used to branch/return to the specified subroutine.

Syntax

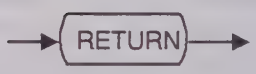
(1)-1



(1)-2

GOSUB <line number|label expression>

(2)-1



(2)-2

RETURN

Description

- Moves the processing control to the subroutine starts with the line number defined with the label expression and returns to the next statement to the GOSUB statement by the RETURN statement.
- Be sure to input the RETURN statement at the end of subroutine and return the processing control to the main program.
- If the RETURN statement is executed without the branch to subroutine, an error may occur.
- Since Nest is available between the GOSUB statement and RETURN statement, the processing can branch to the other subroutine. If more Nest is performed, the remaining capacity (space) of BASIC program will be decreased and then an error may occur.
- If the line of the label expression defined in GOTO/GOSUB does not exist, or if the line of that label expression is deleted by mistake, the value of the label expression in GOTO/GOSUB becomes 0. When it runs as it is above, the following message is displayed.

  Undefined line

  The program is not able to be executed as it is. Correct the line of the GOTO/GOSUB to be the right label expression.

Example

```
10 FOR I=1 TO 9
20   GOSUB 60
30   GOSUB *PRT
40 NEXT I
50 STOP
60 ! SUB ROUTINE
70 X = I * I
80 RETURN
90 *PRT ! SUB ROUTINE
100 PRINT I; " * " ;I; " = " ;X
110 RETURN
```

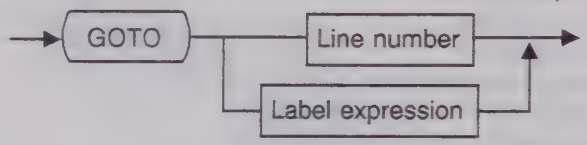
29. GOTO

Outline

The GOTO statement is used to branch to the specified line.

Syntax

(1)-1



(1)-2

GOTO <line number|label expression>

Description

The GOTO statement branches to the specified line number or label line unconditionally.

Example

```
10 FOR I=1 TO 9
20   GOTO 60
30   GOTO *PRT
40 NEXT I
50 STOP
60 !
70 X = I * I
80 GOTO 30
90 *PRT
100 PRINT I; " * " ;I; " = " ;X
110 GOTO 40
```

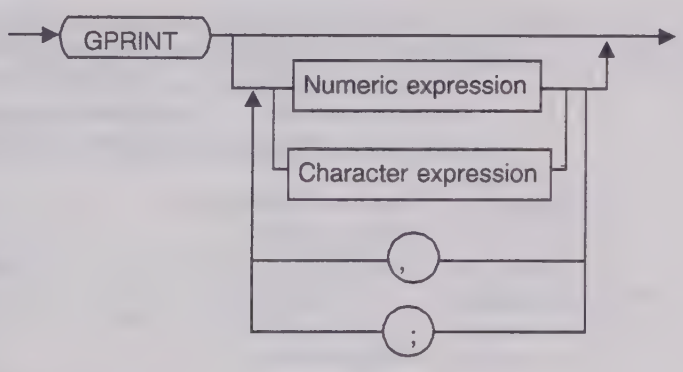
30. GPRINT

Outline

This statement is used to output numerics or character strings.
GPRINT: GPIB output

Syntax

(1)-1



(1)-2

GPRINT [<numeric expression|character expression> {, | <numeric expression|character expression>}

Description

- This statement displays the numerics or character strings specified by the GPRINT.
- When the multiple numerics or character strings are delimited with a comma and specified, they are continuously output without LF.
- If a semicolon is used at the end of the GPRINT statement, LF could not be performed after the termination of print out. Therefore, if the next GPRINT statement is executed, the line followed after the previous output line will be output continuously.

< Sophisticated usage >
It is possible to load the data of the variable used in the BASIC from the external controller (a personal computer and so on) when this analyzer is in the ADDRESSABLE mode (CONTROL 7:0).

Example: To load the data of the variable Aa, Bb$. (The GPIB address of this analyzer is supposed to be configured at 8.)
The current variable is supposed to be Aa = 123.456 and Bb$ = "ADVANTEST". The following commands are executed from the external controller. (In the case of using NEC-PC98 N88BASIC.)

Example 1

```
PRINT @8;''@GPRINT AA''
INPUT @8;ENT1
PRINT EXT1          .........  Execution result: 123.456
```

Example 2

```
PRINT @8;''@GPRINT AA,BB$''
INPUT @8;ENT2$
PRINT EXT2$         .......  Execution result:
                             123.456 ADVANTEST
```

Note: When two or more variables are specified after the GPRINT statement (example 2), the expected value is not assigned to the second and later variables even if the loading is performed at the multiple variables on the side of the external controller. That is because the GPRINT statement outputs it to GPIB as one data of the character string.

Example

```
100 PRINTER 1
110 FOR I=0 TO 20
120   GPRINT I
130 NEXT I
140 STOP
```

31. IF-THEN, ELSE, END IF

Outline

This statement is used to perform the branch based on the condition branch and the specified statement.

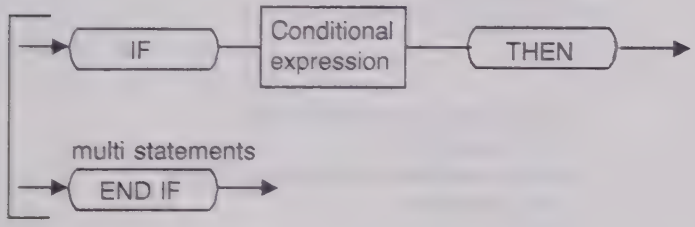
Syntax

(1)-1

(1)-2

IF conditional expression THEN statement

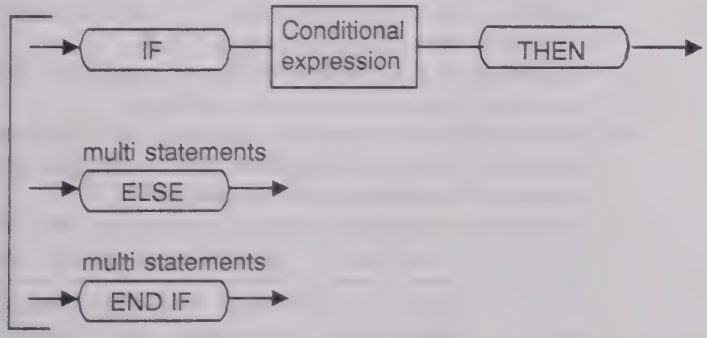
(2)-1



(2)-2

```
IF conditional expression THEN
  multi statements
END IF
```

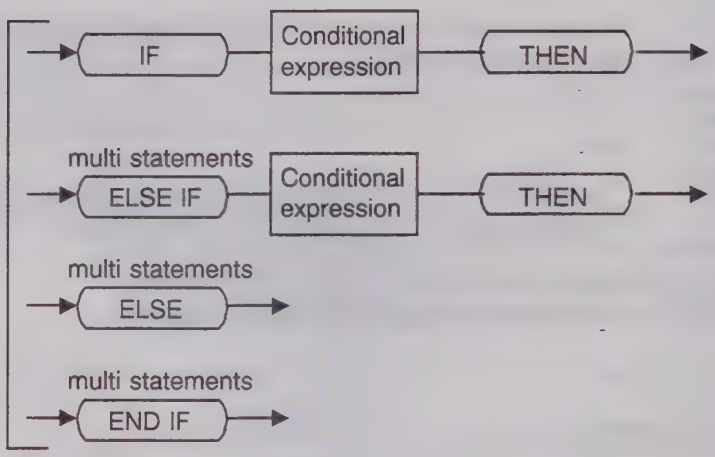
(3)-1



(3)-2

```
IF conditional expression THEN
  multi statements
ELSE
  multi statements
END IF
```

(4)-1



(4)-2

```
IF conditional expression THEN
  multi statements
ELSE IF conditional expression THEN
  multi statements
ELSE
  multi statements
END IF
```

Description

- Generally, the condition expression represents a logical expression, however, numeric expression can be used in this statement other than the logical expression used relational operators. In this case, when the calculation result becomes 0 only, the value is determined as FALSE, and the values other 0 is estimated as TRUE.
- Depending on the condition of logical expression, branching and processing the program can be performed.
- When the logical expression is defined, the THEN statement can be executed. The other statements can be followed after the THEN statement and the next statement can be executed.
- If the logical expression cannot be concluded, the next line is performed.
- The following six types of relational operators are provided:

A = B	Returns true if A equal to B; false otherwise.
A > B	Returns true if A is greater than B; false otherwise.
A < B	Returns true if A is less than B; false otherwise.
A > = B	Returns true if A is greater than or equal to B; false otherwise.
A < = B	Returns true if A is less than or equal to B; false otherwise.
A < > B	Returns true if A does not equal to B; false otherwise.

In the logical expression above, both values A and B consist of numeric expression. The comparison between numeric expression and character-string expression can be performed.

Example

```
10 FLG = 0
20 FOR I=0 TO 10
30  PRINT I;
40  IF (I % 2) =0 THEN FLG = 1
50  IF FLG = 1 THEN
60                    PRINT "  EVEN" ;
70                    FLG = 0
80                    END IF
90  PRINT
100 NEXT I
110 STOP
```

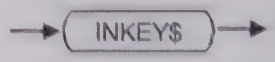
32. INKEY$

Outline

The INKEY$ statement is used to return the cord of the panel key just before pressed. And the data of this variable is cleared after it is referred once.

Syntax

(1)-1



(1)-2

INKEY$ →

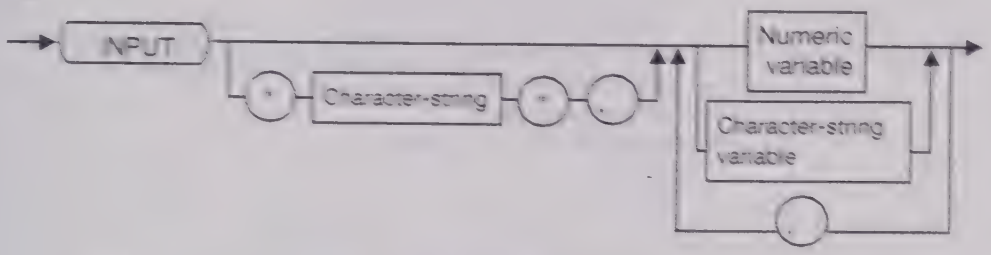
Description

PRINT INKEY$ → Loads the key cord just before input. (Result: A)
Key$ = INKEY$ → Loads the key cord just before input.

33. INPUT (INP)

Outline The INPUT statement is used to assign the data entered by keys to numeric variables.

Syntax (1)-1



(1)-2

INPUT {"character-string";} <numeric variable | character-string variable> {,numeric variable | character-string variable}

Description

```
INPUT A                          →Assigns the data entered by keys to the numeric
                                  variable A.
INPUT A, B, C                    →Assigns the data entered by keys to the numeric
                                  variables A, B, C.
INPUT "Please input", A$         →Displays 'Please input' onto the screen and
                                  assigns the data entered by keys to the
                                  character string variable A$.
```

Note: When the INPUT statement is executed, then the program waits for next key input from the panel or the keyboard and the waiting state for the key input is continued until the ENTER key is pressed after the data is input.

Note

- When two or more variables are specified, the data to be input should be input for the number of variables that are delimited with a comma and are specified.
- When the characters other than the numeric (such as alphabets and signs and so on) are entered during the waiting state of the program in case of numeric variable input, then they will be ignored. If no numeric is existed, then 0 will be assigned to the numeric variable.
- When the character constant is entered during the waiting state in case of character string variable input, it need not be enclosed in the double quotation (").
- If there are no numeric variable and character string and only the ENTER key is pressed, no assignment to the variable is performed. In other word, the value immediately before the INPUT statement ha been remaining.

Example

```
10 OUTPUT 31;"IP"
20 INPUT "CENTER FREQUENCY(MHz)?",Cf
30 INPUT "SPAN FREQUENCY(KHz)?" ,Sf
40 OUTPUT 31;"CF",Cf,"MZ"
50 OUTPUT 31;"SP",Sf,"KZ"
60 OUTPUT 31;"TS"
70 OUTPUT 31;"PS"
80 OUTPUT 31;"ML?"
90 ENTER 31;Mkr$
100 PRINT "LEVEL =",Mkr$
110 STOP
```

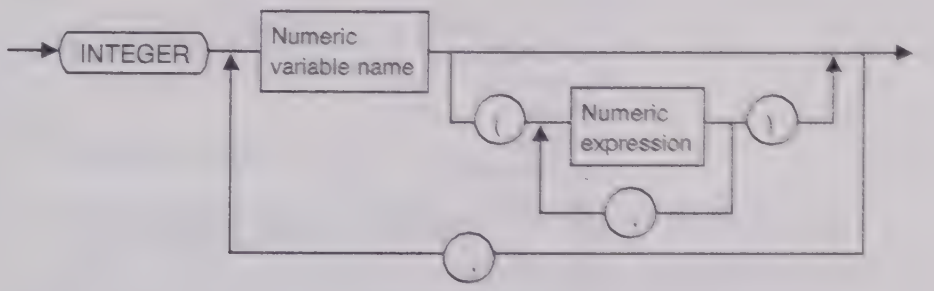
34. INTEGER

Outline

The INTEGER statement is used to declare that the variable or array variable is an integer type.

Syntax

(1)-1



(1)-2

INTEGER A[B] {, A[B] }

A: Numeric variable name
B: (Numeric expression {, Numeric expression})

Description

- When a numeric variable or an array variable is specified in the INTEGER statement, the variable is determined as an integer type after the specification.
- The numeric handled in the integer-type variable, it is the same as the range of an integer constant.
  -2147483648 to +2147483647
- In the variables which handle only the integers, the declaration in the INTEGER statement is recommended to shorten the processing time.
- When the array declaration is used in the INTEGER statement, the specified-size array variable is reserved on the memory. If larger array declaration is performed, an error may occur due to the rack of memory space (memory space full) and then the program execution will be forcibly terminated.
  (Memory space full)
- When multiple subscripts are specified, the array variables are also specified according to the number of dimension. (The number of dimension that can be specified is 10 at the maximum.)

Example

```
10  INTEGER ARRAY(2,3)
20  PRINT "J/I " ;
30  PRINT USING "X,3D,3D,3D" ;1,2,3
40  PRINT " " ;
50  FOR I = 1 TO 2
60    FOR J = 1 TO 3
70        ARRAY(I,J) = I*10 + J
80    NEXT J
90  NEXT I
100 FOR I = I TO 2
110 PRINT
120 PRINT USING  " 2D,2X,# " ;I
130   FOR J = 1 TO 3
140       PRINT USING  "3D,#" ;ARRAY(I,J)
150   NEXT J
160 NEXT I

<Result>
 J/I  1  2  3

  1  11 12 13
  2  21 22 23
```

CAUTION

- The variable which is once specified as an integer type by the INTEGER statement, if the instruction is deleted by the DEL or comment statement, the specified variable (integer type) is not changed.
- To change the specified integer-type variable into a real-type variable again, add the DIM instruction or execute the SAVE/LOAD command once and then perform the RUN command. (In the case of a numeric variable, it is changed into a real type variable by performing the REAL command.)

35. INTERFACE CLEAR

Outline

The INTERFACE CLEAR statement is used to initialize the all GPIB interfaces connected with the analyzer.

Syntax

(1)-1

(1)-2

INTERFACE CLEAR

Description

- When the INTERFACE CLEAR statement is executed, the GPIB single signal IFC is output approximately 100μs.
  If the all GPIB interface devices connected with the analyzer receive the IFC signal, then the setting state of talker or listener will be canceled.

Example

```
10 INTERFACE CLEAR
```

Note

The INTERFACE CLEAR statement is not available in the ADDRESSABLE mode.

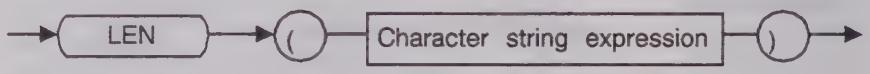
36. LEN

Outline

The LEN is a built-in function that is used to return the length of the character string.

Syntax

(1)-1



(1)-2

LEN (character string expression) →

Example

```
PRINT LEN("ABCDE")
A=LEN(Str$)
```

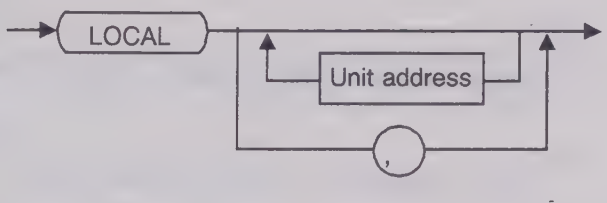
37. LOCAL

Outline

The LOCAL statement is used to cancel the specified device from the remote state or to set the remote-enable (REN) line to FALSE.

Syntax

(1)-1



(1)-2

LOCAL [unit address {, unit address}]

Description

- If only the LOCAL statement is executed without specifying the device address, then the GPIB remote-enable line will become FALSE (High level) and all the devices on the GPIB will be a local state.
  If the REN is FALSE, pay attention that the setting of GPIB device could not be performed (cannot be controlled by GPIB).
- To set the REN to TRUE (Low level) again, execute the REMOTE.
- If the device address is specified followed after the LOCAL, only the device specified by the device address could be addressed, and the remote state will be canceled.

Example

```
10 LOCAL
20 LOCAL 1
30 LOCAL 1,2,3
```

Note

The LOCAL state is not be available in the ADDRESS mode.

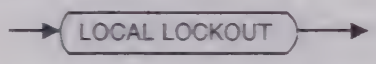
38. LOCAL LOCKOUT

Outline

The LOCAL LOCKOUT statement is used to prohibit the function which controls the local remote state from the panel key of the device connected to the GPIB.

Syntax

(1)-1



(1)-2

LOCAL LOCKOUT

Description

- When each device is remote state (controlled by GPIB), the panel key of each device is locked except for the LOCAL key and the data setting cannot be performed from each panel.
  When the LOCAL key is pressed during the remote state, the data setting is available since each device become local state. Therefore, various errors occur during the remote control and the control cannot be performed correctly.
  In this case, if the LOCAL LOCOUT statement is executed, its function enables to lock the all devices on the GPIB and the setting from each device panel can be completely prohibited.
- When the LOCAL LOCKOUT statement is executed, the local lockout (LLO) of universal command is sent to the GPIB.
- To cancel the local lockout state, use the LOCAL command to set the REN line to FALSE (High level).

Example

```
10 LOCAL LOCKOUT
```

Note

The LOCAL LOCKOUT statement is not available in the ADDRESSABLE mode.

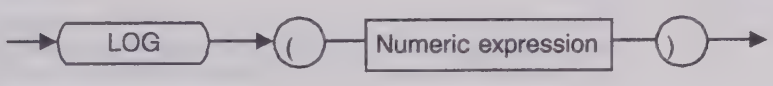
39. LOG

Outline

The LOG is a built-in function that is used to return the value of the natural logarithm.

Syntax

(1)-1



(1)-2

LOG (numeric expression) →

Description

```
PRINT LOG(10)
A=LOG(B)
```

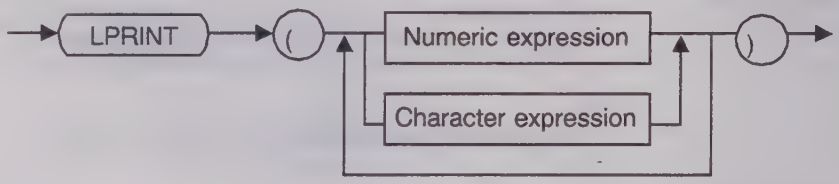
40. LPRINT

Outline

The LPRINT statement is used to output the numeric value or the character string and so on through the parallel port.

Syntax

(1)-1



(1)-2

LPRINT numeric expression | character expression {, | ; numeric expression | character expression}

Description

`LPRINT "ADVAN"` → Outputs the character string ADVAN through the parallel port.

`LPRINT 123.456` → Outputs 123.456 through the parallel port.

`LPRINT A, B, C$` → Outputs the data of the variable A, B, C$ through the parallel port.

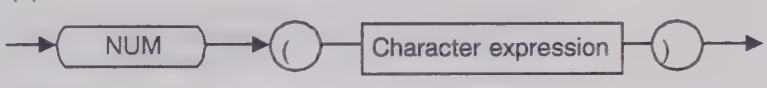
41. NUM

Outline

The NUM is a built-in function that is used to return the ASCII cord of the specified character.

Syntax

(1)-1



(1)-2

NUM (character expression) →

Description

```
PRINT NUM("a")
A=NUM(B$)
```

42. OFF END

Outline

The OFF END statement is used to cancel the processing of the end of file specified by the ON END statement.

Syntax

(1)-1

(1)-2

OFF END # file descriptor

Description

- After canceling the branch defined into file descriptor, if the end of file occurs, the following error message will be displayed and the program will be terminated.

  End of "file name" file

Note: For the information how to handle files, refer to "2.4 File Management".

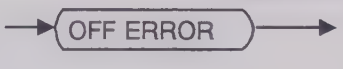
43. OFF ERROR

Outline

The OFF ERROR statement is used to cancel the branch function when an error occurs.

Syntax

(1)-1



(1)-2

OFF ERROR

Description

- The OFF ERROR statement prohibits the error branch defined by the ON ERROR statement.

Example

```
10  ON ERROR GOTO 100
     :
100 OFF ERROR
110 PRINT "Error Code" ,ERRN
120 STOP
```

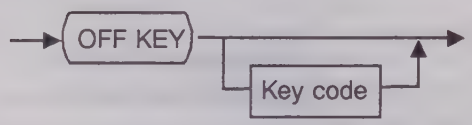
44. OFF KEY

Outline

The OFF KEY statement is used to prohibit the branch by the interruption of the soft keys (1 to 7) on the panel. The display of the buttons on the software key menu on the screen is cleared by executing this command statement. The software key menu itself is also cleared by prohibiting the branch by interruption of all the soft key.

Syntax

(1)-1



(1)-2

OFF KEY [key code]

Description

- The OFF KEY statement prohibits the branch by the interruption of the analyzer KEY input, which is permitted by the ON KEY statement.

Example

```
10  ON KEY 2 GOTO 100
20  ENABLE INTR
30  ! LOOP
40  GOTO 30
100 OFF KEY
110 PRINT "OFF KEY"
120 STOP
```

45. OFF SRQ, OFF ISRQ

Outline

This statement is used to cancel the function and definition by the interruption of SRQ or ISRQ.

Syntax

(1)-1

(1)-2
OFF SRQ

(2)
The OFF ISRQ is the same as the OFF SRQ.

Description

- OFF SRQ
  This statement prohibits the branch by the interruption, which is permitted by the ON SRQ.
- OFF ISRQ
  This statement prohibits the branch by the interruption, which is permitted by the ON ISRQ.

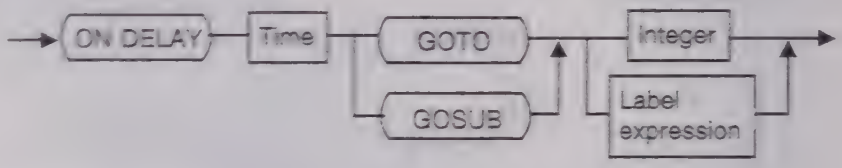
46. ON DELAY

Outline

The ON DELAY statement is used to branch after the specified time elapsed.

Syntax

(1)-1



(1)-2

ON DELAY time <GOTO | GOSUB> <integer | label expression>

Note: The unit of time is msec, and the setting range is between 0 to 65535.

Description

- The ON DELAY statement branches according to the statement after the specified time elapsed.
- Acceptance of the interruption should be permitted by the ENABLE INTR statement.

Example

```
10  INTEGER T
20  T=50
30  ENABLE INTR
40  ON DELAY T GOSUB *TEST
50  STOP
100 *TEST
110  PRINT T;"[msec] Delay"
120 RETURN
```

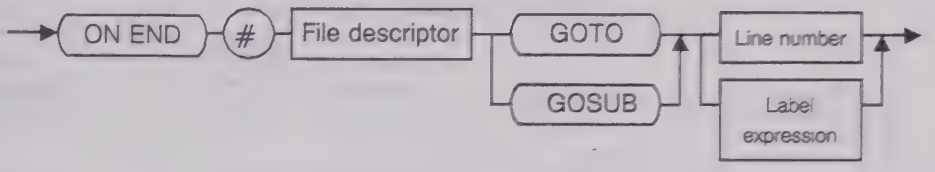
47. ON END

Outline

The ON END statement is used to define the processing (destination branch) at the end of file.

Syntax

(1)-1



(1)-2

ON END #file descriptor <GOTO | GOSUB> <line number | label expression>

Description

- The ON END statement reads out the data from the file by the ENTER command, if the data to be entered is not existed with reading out the end of file, the result will be the end of file.
  If the processing declaration is omitted in the ON END statement, after closing the file, an error message will be displayed and the program will terminate.
- The branch destination is specified with the numeric variable, the numeric constant or the label.

Note: For the information how to handle files, refer to "2.4 File Management".

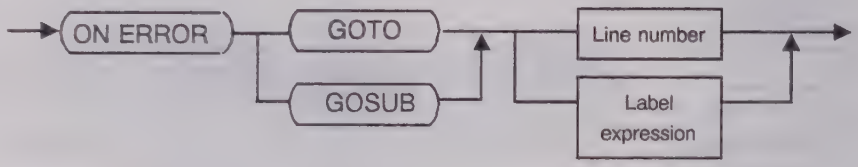
48. ON ERROR

Outline

The ON ERROR statement is used to permit the branch when an error occurs.

Syntax

(1)-1



(1)-2

ON ERROR <GOTO | GOSUB> <line number | label expression>

Description

- If an error occurs during the BASIC program, the statement number and error message of the program will be displayed and the program will terminate.
  Especially, if the built-in function error which demands the service request of the measuring device, only the error message will be displayed and the program will continue the operation. To detect the error to branch, use the ON ERROR statement is used.
- The branch destination is specified with the numeric constant, the numeric variable or the label.
  To categorize the generated error, the ERRN system variable which stores the error number is provided.
- After generating the error, if the error is not recovered by the error processing, then the endless loop will be performed. To prevent this trouble, the OFF ERROR statement must be used (written).

Example

```
ON ERROR GOTO 1000
```

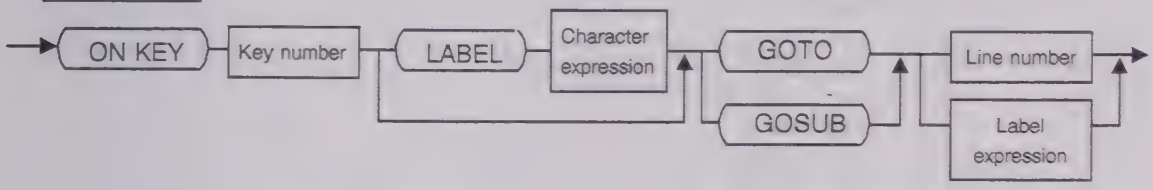
49. ON KEY

Outline

The ON KEY statement is used to permit the branch by the interruption of panel softkey (1to7) input.
Any label characters of every softkey can be displayed onto the software key menu of the display by using this command statement.

Syntax

(1))1



(1)-2

ON KEY <key number> [LABEL character-string expression] <GOTO | GOSUB> <line number | label expression>

Description

- Available Range of the key number: 1 to 7

```
ON KEY 1 GOTO 100      →Goes to the process of line number 100 by
                         pressing the soft key 1.
ON KEY 2 GOTO *Lb1     →Goes to the process of the line with label *Lb1 by
                         pressing the soft key 2.
ON KEY 7 LABEL "Soft-key | 7" GOSUB *Sub
                       →Pressing the soft key 7, *Sub function is called by
                         a subroutine.
```

Note

- Acceptance of the interruption should be permitted by the ENABLE INTR statement to make this statement enable.
- The character string expression continued to the LABEL statement can be specified by (ten-characters x three-lines). And if a pipe (|) is specified in the character string, a new line is started there in the characters displayed in the button.
  (Two pipes can be written in the character string at the maximum. When three or more pipes are written, the characters after the third pipe will be truncated.)

Example

The process is allocated to the soft keys 1 to 3.

```
10   ON KEY 1 GOTO *key1
20   ON KEY 2 LABEL ''Softkey-2'' GOSUB *key2
30   ON KEY 3 LABEL ''Softkey | key2'' GOSUB *key3
40   ENABLE INTR
50   GOTO 50
60   !
70   *key1
80   PRINT ''Pressed'' ;inkey$
90   GOTO 40
100  *key2
110   PRINT ''Softkey-2 pressed!!''
120  RETURN
130  *key3
140  STOP
```

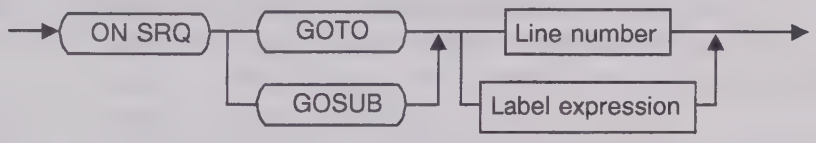
50. ON SRQ, ON ISRQ

Outline

The ON SRQ statement is used to permit the interruption branch by the GPIB external SRQ signal. (It is available in ON SRQ controller mode only.)
The ON ISRQ statement is used to permit the interruption branch when the internal interruption factor is generated.

Syntax

(1)-1



(1)-2

ON SRQ <GOTO | GOSUB> <line number | label expression>

(2)

The ON ISRQ is the same as the ON SRQ.

Description

- This statement branches by the interruption during the program execution.
- The branch is executed after completing the processing of the statement being executed when the interruption is generated.
- The return position of the statement when the program branches to the subroutine is the next statement of the statement being executed when the interruption is generated.
- The ON SRQ statement performs the interruption branch by the SRQ signal from the GPIB external during the controller mode in progress.
- Acceptance of the interruption should be permitted by the ENABLE INTR statement.

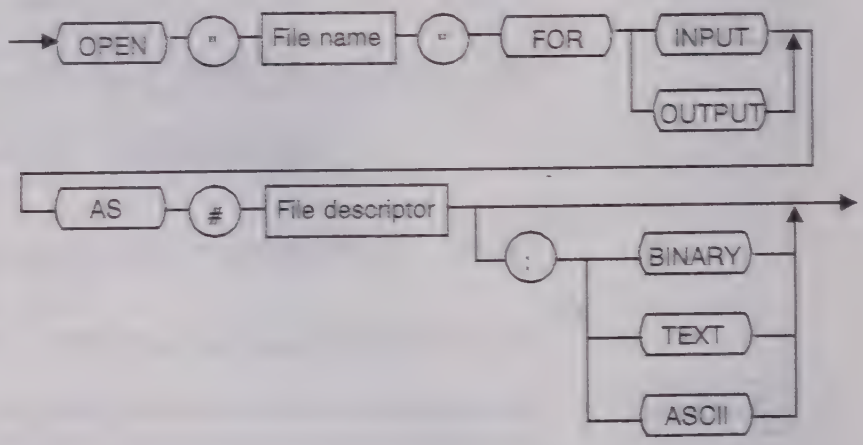
51. OPEN

Outline

The OPEN statement is used to assign the file descriptor to the file and to open the by with the specified processing mode.

Syntax

(1)-1



(1)-2

OPEN "file name" FOR processing mode AS #file descriptor [; file type]

Note: Processing mode: INPUT | OUTPUT
File type: BINARY | TEXT | ASCII

Description

- To recognize the file for the program, the OPEN statement assigns the file descriptor to the file and to open the by with the specified processing mode.

  Processing mode
  Two processing modes are provided.
  OUTPUT: Used for writing the data to files.
  INPUT: Used for reading out the data from files.

  # File descriptor
  Generally, writing reading files uses the ENTER or OUTPUT mode.
  For these commands, the file descriptor is used to recognize the target files. To name the file descriptor, use alphanumerics followed after #.

File type

Three file types (BINARY, TEXT, and ASCII) are provided.

If the file type is not specified, BINARY type is automatically set.

BINARY: Stores the data without changes. An integer type is 4-byte data, a real type for 8-byte data, and a character-string type for header 4-byte. In case of the character-string type, ASCII data is followed after the header 4-byte. If the number of character data is an odd, then one space of 1-byte will be followed after the data.

TEXT: Converts data into ASCII codes and outputs the data, and "-" or space is followed before the numeric. The USING specification can be used for the TEXT file.

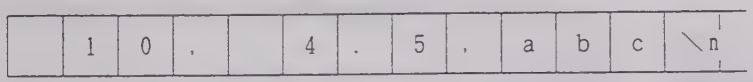
ASCII: Represents the input/output item using ASCII codes followed after 2-byte header. "-" or space is followed before the numeric. If the number of the character data is an even, then one space will be followed after the data.

- When the file descriptor already assigned the file to the other file is opened, the previous assigned file is closed and the specified file is newly opened.
- The same files cannot be opened using the multiple file-descriptor at the same time.

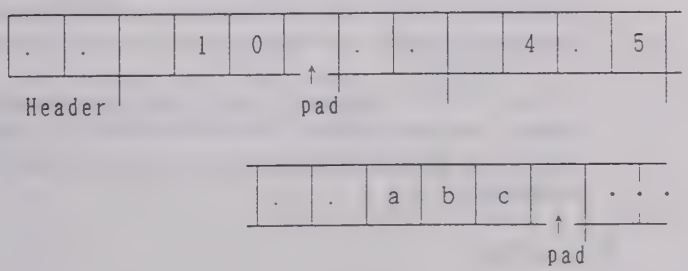
Note: For the information how to handle files, refer to "2.4 File Management".

Example

```
10 OPEN "DATA.BAS" FOR OUTPUT AS #FD ; TEXT
20 OUTPUT #FD;10,4.5,"abc"
```




```
10 OPEN "DATA.BAS" FOR OUTPUT AS #FD ; ASCII
20 OUTPUT #FD;10,4.5,"abc"
```



52. OUTPUT

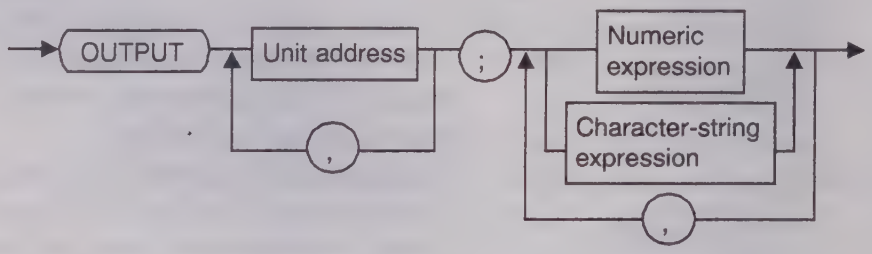
Outline

(1) The OUTPUT statement is used to output the data to GPIB.

(2) The OUTPUT statement is used to output (write) the data to files.

Syntax

(1)-1

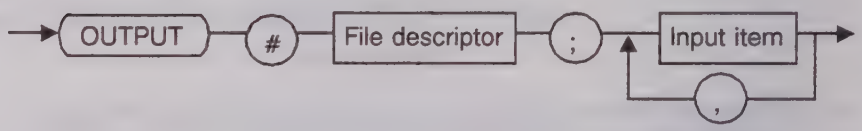


(1)-2

OUTPUT unit address {, unit address} ; <numeric expression | character-string expression> {, <numeric expression | character-string expression>}

Note: Unit address: 0 to 30; Address of the external GPIB device.
31; Output to the measurement section of the analyzer.

(2)-1



(2)-2

OUTPUT # file descriptor ; input item {, input item}

Description

Syntax of (1)

- The OUTPUT statement sends numeric and character string as an ASCII data to the specified device by the unit address.
  Multiple unit address can be specified by delimiting with a string of commas. The numeric expression and the character-string expression are used together by delimiting with a string of commas.

- If the OUTPUT statement is executed when the REN line is TRUE (Low level), the unit specified by the unit address will be automatically remote state. To cancel the remote state by the program, execute the LOCAL statement.

- Example

```
10 A=5
20 B=10
30 OUTPUT A;"STARTF", B,"MHz"
```

- Note
  In the SYSTEM CONTROLLER mode, the specified address device is set as the listener and the data is output.
  When the external listener is not existed, this command cannot be executed.

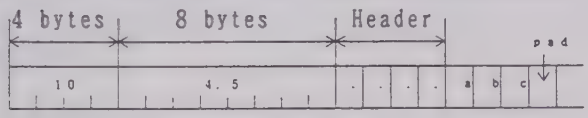
Syntax of (2)

- The OUTPUT statement converts the data into the BASIC format and then outputs the file assigned to the file descriptor.
  The OUTPUT statement reads out the converted BASIC-format data and assigns it to its input item.

- Example 1: BINARY file
  Outputs data without changes. A character string is output with the header which indicates the length of 4-byte character string. If the number of character data is an odd, then one space of 1-byte will be followed after the data.

```
10 OPEN "FILE" FOR OUTPUT AS #FD
20 OUTPUT #FD;10,4.5,"abc"
```

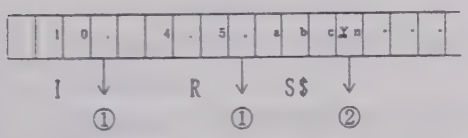
Note: For the information how to handle files, refer to "2.4 File Management".



Header has each data length.

- Example 2: TEXT file
  Converts data into into ASCII codes and outputs the data.
  The signs (space or minus) for numeric data is placed to the top of the field.

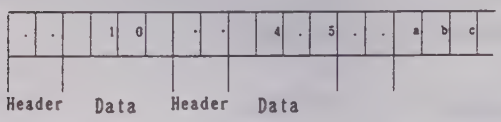
```
10 OPEN "FILE" FOR OUTPUT AS #FD;TEXT
20 OUTPUT #FD;10,4.5,"abc"
```



①: Each item is delimited with a string of commas.
②: LF followed after the final item is output.

- Example 3: ASCII file
  Converts data into ASCII codes and outputs the data.
  The signs (space or minus) for numeric data is placed to the top of the field. If the number of character data is an odd, then one space of 1-byte will be followed after the data.

```
10 OPEN "FILE" FOR INPUT #FD;ASCII
20 OUTPUT #FD;10,4.5,"abc"
```



Header has each data length.

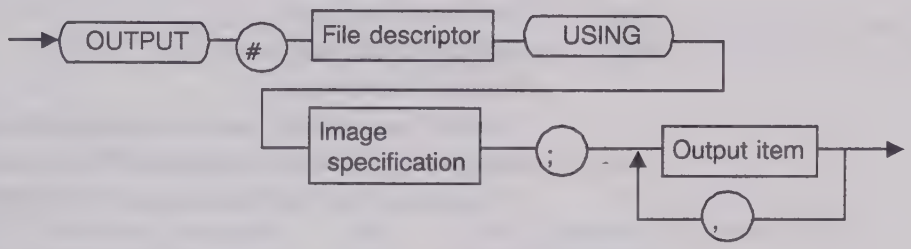
53. OUTPUT USING

Outline

The OUTPUT USING statement is used to output data with the specified data-type to the file assigned to the #file descriptor.

Syntax

(1)-1



(1)-2

OUTPUT # file descriptor USING image specification ; output item {, output item}

Note: OUT can be used instead of the OUTPUT, and USE for the USING.

Description

- When the USING and the image specification are specified, the format is converted and output. The image specification must be specified by character-string expression.
- The specified file descriptor when the file is opened is used. The file descriptor is assigned for the file to be objected at the file open. After that, the processing for the file can be performed through this file descriptor.

image specification

D: Specifies the output digits with No. of D. A space is used to fill up the remaining blank in the specified field.

Z: Specifies the output digits with No. of Z. A zero is used to fill up the remaining blank in the specified field.

K: Displays the expression as it is.

S: Displays the OUTPUT USING with a + or - sign flag at the position of S.

M: Displays the OUTPUT USING with a - for negative and a space for positive at the position of M.

.: Displays the OUTPUT USING to match the position "." with coming the decimal point.

E: Displays OUTPUT USING with the exponent format (e, sign, exponent).

H: Same as K. However, use a comma for a decimal point.

R: Same as ".". However, use a comma for a decimal point.

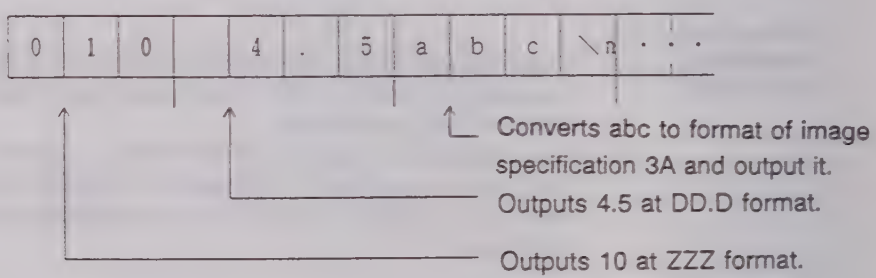
*: Specifies the output digit with the number of *. A space is used to fill up the remaining blank in the specified field.

A:	Displays one character.
k:	Displays the character-string expression as it is.
Literal:	Encloses the literal with \" when writing it in the format expression.
X:	Displays the character of one space.
B:	Displays the expression result using an ASCII code.
@:	Outputs the form lead.
+:	Outputs the carriage return.
-:	Outputs the line feed.
#:	Does not hang the line feed immediately followed after the last item.
n:	Specifies the number of repetition of each image by using numerics. For example, 3D.2D is the same as for DDD.DD, and 4A for AAAA.

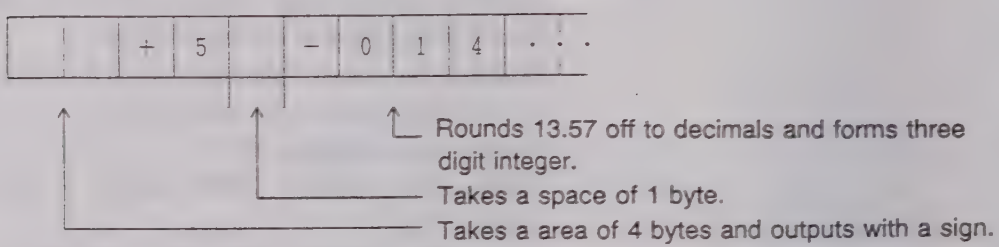
Note: For the information how to handle files, refer to "2.4 File Management".

Example

```
OUTPUT #FD USING "ZZZ,DD.D,3A";10,4.5,"abc"
```


0 1 0 4 . 5 a b c \n ・・・
Converts abc to format of image specification 3A and output it.
Outputs 4.5 at DD.D format.
Outputs 10 at ZZZ format.

```
OUTPUT #FD USING "SDDD,X,MZZZ";+5,-13.57
```


+ 5 − 0 1 4 ・・・
Rounds 13.57 off to decimals and forms three digit integer.
Takes a space of 1 byte.
Takes a area of 4 bytes and outputs with a sign.

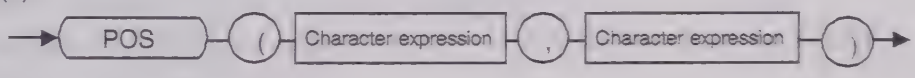
54. POS

Outline

The POS is a built-in function that is used to search the character string for the specified character and return its position detected (where from the initial character).

Syntax

(1)-1



(1)-2

POS (character expression, character expression)

Description

```
PRINT POS(''ADVANTEST'',''VA'')
A=POS(A$, B$)
```

→'VA' is detected at the third character, so 3 is returned.

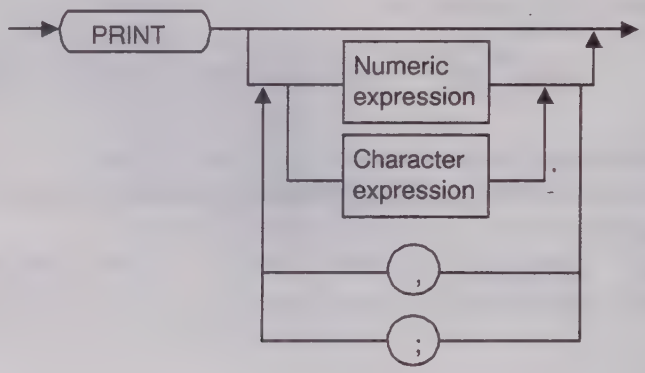
55. PRINT [USING]

Outline

The PRINT [USING] statement is used to display numerics or character strings.

Syntax

(1)-1



(1)-2

PRINT [numeric expression| character expression {, | ; numeric expression| character expression}]

Description

- The PRINT [USING] statement displays the specified numeric or character string.
- When the multiple numerics or character strings are delimited with a comma and specified, they are continuously output without LF.
- If a semicolon is used at the end of the PRINT statement, LF could not be performed after the termination of print out. Therefore, if the next PRINT statement is executed, the line followed after the previous output line will be output continuously.

Example

```
10 PRINT 123*456
20 PRINT "ABC"
30 PRINT "Freq.=",A, "Hz"
40 PRINT I,
```

- In PRINT USING format specification expression : [[expression [--]]

  The format specification expression (character-string expression), specify the image specification by using a comma among image. The end of the format specification expression is automatically returned with line feed.

  image specifications

  D: Specifies the output digits with No. of D. A space is used to fill up the remaining blank in the specified field.

  Z: Specifies the output digits with No. of Z. A zero is used to fill up the remaining blank in the specified field.

  K: Displays the expression as it is.

  S: Displays the PRINT USING format with a + or - sign flag at the position of S.

  M: Displays the PRINT USING format with a - for negative and a space for positive at the position of M.

  .: Displays the PRINT USING format to match the position "." with coming the decimal point.

  E: Displays PRINT USING format with the exponent format (e. sign. exponent).

  H: Same as K. However, use a comma for a decimal point.

  R: Same as ".". However, use a comma for a decimal point.

  *: Specifies the output digits with the number of *. A space is used to fill up the remaining blank in the specified field.

  A: Displays one character.

  k: Displays the character-string expression as it is.

  X: Displays the character of one space.

  Literal: Encloses a literal with " when writing it to the format expression.

  B: Displays the expression result using an ASCII code.

  @: Form lead

  +: Moves the display position to the top of the same line.

  -: Line feed

  #: Does not line feed.

  n: Specifies the number of repetition of each image by using numerics. For example. 3D.2D is the same as for DDD.DD. and 4A for AAAA.

Example 1

```
10 PRINT USING "4Z,2X,5D,2X,5*" ;123,-444,567

<After the execution>
 0123   -444   **567
```

Example 2

```
10 PRINT USING "S3D,X,S3D" ;-4.5,465
20 PRINT USING "M3Z.Z,X,M3ZR3Z" ;1.26,-5.452

<After the execution>
  -5 +456
 001.3 -005.452
```

Example 3

```
10 PRINT USING "K,X,H" ;5.03884e+22,4.5563

<After the execution>
 5.03884e+22 4.5563
```

Example 4

```
10 PRINT USING "k,#" ;"character:"
20 PRINT USING "B" ;69

<After the execution>
 character:E
```

Example 5

```
10 PRINT USING "\" ..... \" ,+,A" ; "*"
20 PRINT USING "k,-, \" .END. \" " ;  "string"

<After the execution>
 *.....
 string
 .END.
```

Example 6

```
                                    <After the execution>
100 PRINT USING "DDD.DD" ;1.2         1.20
110 PRINT USING "ZZZ.ZZ" ;1.2        001.20
120 PRINT USING "K" ;1.2             1.2
130 PRINT USING "SDDD.DD" ;1.2        +1.20
140 PRINT USING "MDDD.DD" ;1.2        1.20
150 PRINT USING "MDDD.DD" ;-1.2      -1.20
160 PRINT USING "H" ; 1.2            1,2
170 PRINT USING "DDDRDD" ; 1.2        1,20
180 PRINT USING "***.**" ; 1.2       **1.20
190 PRINT USING "A" ; "a"            a
200 PRINT USING "k" ; "string"       string
210 PRINT USING "B" ; 42             *
220 PRINT USING "3D.2D" ;1.2          1.20
```

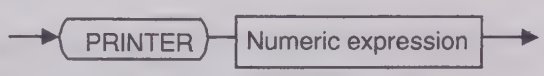
56. PRINTER

Outline

The PRINTER statement is used to specify the unit address for sending the data to the printer.

Syntax

(1)-1



(1)-2

PRINTER numeric expression

Description

- The PRINTER statement sets the printer unit address connected to the GPIB.
- Be sure to specify the printer unit address to the analyzer by the PRINTER statement before executing the PRINT statement.
- The unit address is the integers from 0 to 30.

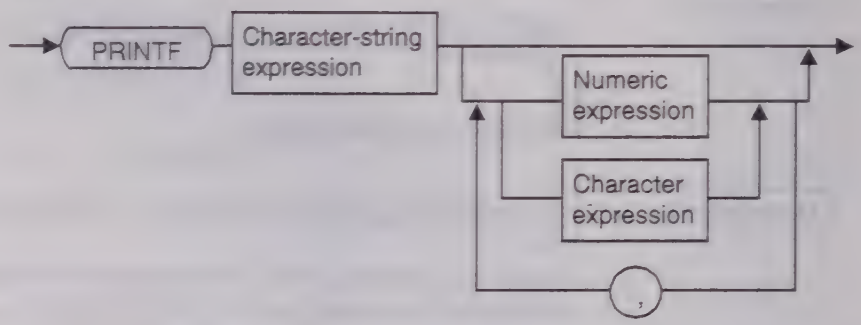
Example

```
10 PRINTER 1
```

57. PRINTF

Outline The PRINTF statement is used to display numerics or character strings.

Syntax (1)-1



(1)-2

PRINTF character-string expression [numeric expression | character expression {, numeric expression | character expression}]

Description

- The PRINTF statement displays the specified numeric or character string.
- When the multiple numerics or character strings are delimited with a comma and specified, they are continuously output without LF. To line feed, use a "¥n" in the format specification expression.
- The first parameter character-string expression is used to specify the preceding parameter format.
- The following format specification are provided.

- PRINTF format specification expression ; [[expression [expression [···]]]

The method of format specification is similarly to the Printf function of C language. The format specification expression is a character-string type and the output format is defined by the following method. The character string other than this format is normally output. If "%" is necessary, add "%" immediately followed after the "%".

%[-] [0] [m] [. n] character

-: Justifies the character with no space from left (if no specification, then from right).

0: Sets the character, which is justified for the remaining blank in the specified field, to be 0.

m: Reserves the field for the character "m".

.n: Outputs the PRINT USING format with n-digit accuracy. In character string, this setup value is used for an actual character-string length.

Character:
- d; decimal with sign
- o; octal
- x; hexadecimal
- s; character string
- e; floating-point expression (exponent format)
- f; floating-point expression

Example

```
10 N = 500000
20 U = LOG(1+1/N)
30 V = U - 1 / N
40 PRINTF  "%7d %12.5e %16.8e \n" ,N,U,V
50 PRINTF  "%s\n" , "end"

<After the execution>
  500000 2.00000e-06 -1.99993982e-12
 end
```

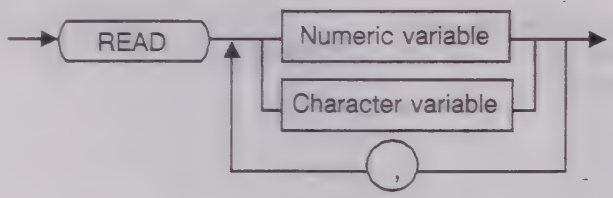
58. READ

Outline

The READ statement is used to assign the constant in the DATA statement to the variable.

Syntax

(1)-1



(1)-2

READ <numeric variable | character variable> {, numeric variable | character variable}

Description

- The READ statement reads the numeric or character string defined in the DATA statement to the variable specified by the argument.
- The READ statement catches the READ statement and searches the DATA statement in the program.
- In the first READ statement, basically (it must be changed by RESTORE statement), the READ searches the constant value from top line to final line in order, and the first searched value is assigned to the variable. After that, the constant corresponding to the DATA statement is searched and assigned to the variable.
- If the constant value specified the DATA statement is less, an error will occur.
- The subjects are the number of the variables read by the READ statement and of the constants of the corresponding DATA statement. It is not concerned that the number of the lines of the DATA statement or of the READ statement.

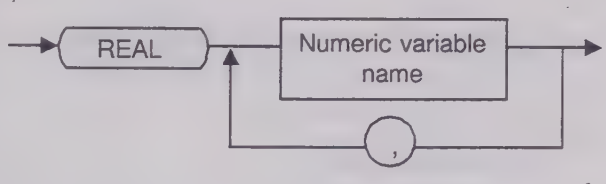
59. REAL

Outline

The REAL statement is used to declare that the numeric variable is the integer type.

Syntax

(1)-1



(1)-2

REAL <numeric variable name> {, numeric variable name}

Description

REAL A, B → Declares that the numeric variable A and B are the integer type.

Note: The variable used when the REAL statement is omitted becomes the integer type.

Note

- The variable declared in the integer type by the INTEGER statement maintains its type until it is changed clearly by the REAL statement so as to be the real number type.

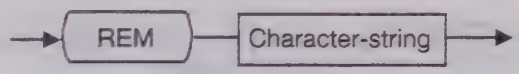
60. REM

Outline

The REM statement is an annotation for program.

Syntax

(1)-1



(1)-2

REM character-string

Description

- The REM statement is used to add the annotation to the program.
- Since the REM statement is no execution statement, any character string can be used followed after the REM statement. All the characters, numerics, and symbols can be used.
- An exclamation mark may be used instead of the REM statement.
- Multi statements using colons followed after the REM statement cannot be used. All the statements are determined as annotation statement.

Example

```
10 REM "PROGRAM 1"
20 ! 1983-JUN-02
30 A=A+1:! INCREMENT A
```

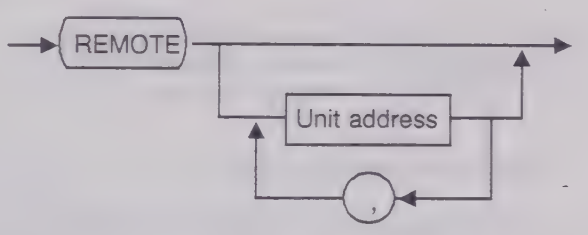
61. REMOTE

Outline

The REMOTE statement is used to set the specified unit to the remote state or to set the remote enable (REN) line to TRUE.

Syntax

(1)-1



(1)-2

REMOTE [unit address {, unit address }]

Description

- If only the REMOTE statement is executed without specifying the unit address, the remote enable (REN) line of the GPIB will become TRUE (Low level) and the unit connected on the GPIB will be set to the remote-controlled state. To set the REN line to FALSE (High level), execute the LOCAL statement.
- If the unit address followed after the REMOTE statement is specified, only the unit address specified by its unit address will be set to the remote-controlled state (only when the REN line is TRUE).
  Multiple unit addresses can be specified.
  To cancel the remote-controlled state, execute the LOCAL statement.
- The REMOTE statement is used to set the selected unit to the remote-controlled state, however, if the following statements are executed, then the specified unit will be automatically set to the remote-controlled state without executing the REMOTE statement.

  CLEAR [unit address {, unit address}]
  OUTPUT unit address {, unit address} ; <output data> {, <output data> }
  REMOTE [unit address {, unit address}]
  SEND LISTEN unit address {, unit address}
  TRIGGER unit address {, unit address}

Example

```
10 REMOTE 1
20 REMOTE 5
30 REMOTE 1 2 3
```

Note

The REMOTE statement is not available in the ADDRESSABLE mode.

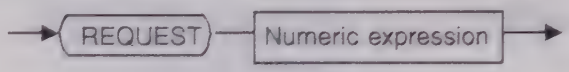
62. REQUEST

Outline

The REQUEST statement is used to set the status byte which is sent to the external GPIB controller in the ADDRESSABLE mode.

Syntax

(1)-1



(1)-2

REQUEST numeric expression

Note: The setting range of integer is between 0 to 255.

Description

- The REQUEST statement sets the status byte which is sent to the external GPIB controller in the ADDRESSABLE mode.
- When the service request (SRQ) is transmitted, the values of 64 to 127 or 192 to 255 (bit 6 indicates "1") must be set.

Example

```
10 REQUEST 65
```

Note

- The REQUEST statement is not available in the SYSTEM CONTROLLER mode.
- Note that the serial pole is used to read (check) ?>the request signal<? from an external controller. The STB? of the GPIB command cannot be used.
- When the SRQD of the GPIB command is executed, the bit 6 of the status byte is always transmitted with "0". Therefore, the SRQ is not transmitted.

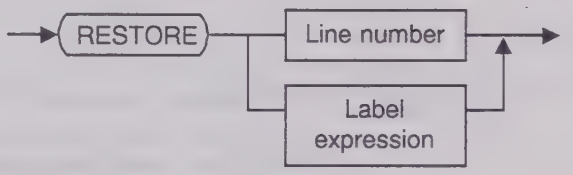
63. RESTORE

Outline

The RESTORE statement is used to specify the DATA line which is read out in the next READ statement.

Syntax

(1)-1



(1)-2

RESTORE <line number | label expression>

Description

- The line number is specified by the numeric expression or label expression. Unless otherwise specified, the constant of the DATA statement is read out from the first line of the program in order, and the DATA statement which is objected for the next READ statement in the RESTORE statement.
- The line number of the argument is the first line number from which the DATA statement search is to start. Therefore, the DATA statement to be specified may be written on the line from which the DATA statement search is to start or any subsequent line.

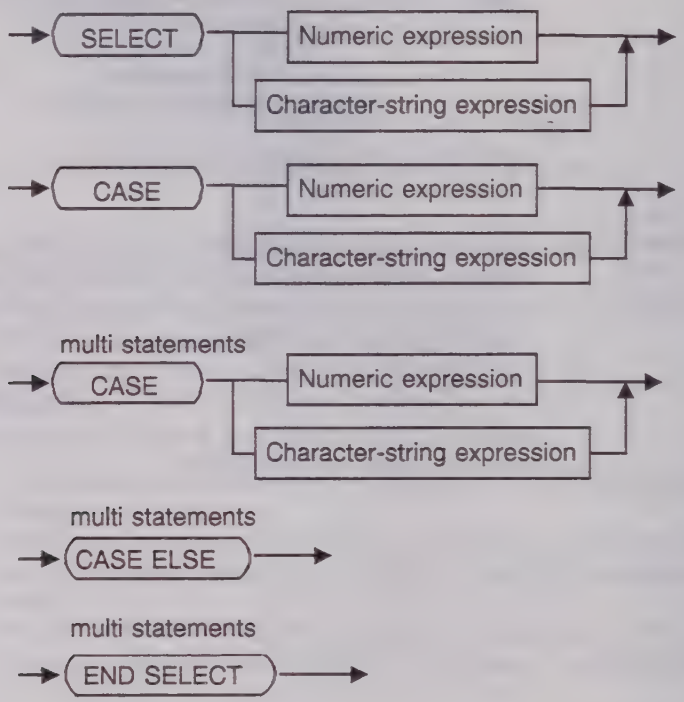
64. SELECT, CASE, ENS SELECT

Outline

This statement is used to perform the multiple brunches on condition of the one expression value.

Syntax

(1)-1



(1)-2

SELECT <numeric expression | character-string expression>
CASE <numeric expression | character-string expression>
multi statements
CASE <numeric expression | character-string expression>
multi statements
CASE ELSE
multi statements
END SELECT

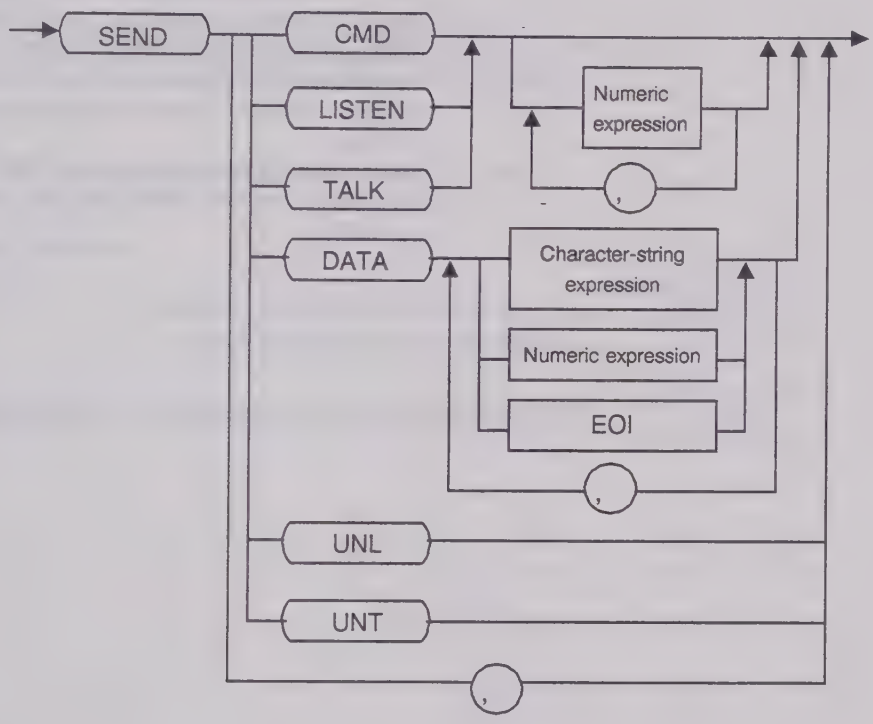
Description

- This statement executes the multiple statements which are agreed with the expression value specified by the SELECT statement followed after the CASE statement.
  The next statements such as CASE, CASE ELSE, or END SELECT can be objected for the execution.
- Nesting can be preformed in the SELECT statement. In this case, an internal SELECT statement includes the other statements.

65. SEND

Outline The SEND statement is used to output the command and data to a GPIB.

Syntax (1)-1



(1)-2

SEND <type A | type B | type C> { <, type A | type B | type C>}

Type A: <CMD | DATA | LISTEN | TALK> [numeric expression {, numeric expression}]
Type B: <DATA> [numeric expression | character string expression {, numeric expression | character string expression}]
Type C: <UNL | UNT>

Description

- The SEND statement sends (transmits) the universal command, the address command, and the data independently to the GPIB.

CMD: Sets the ATN line to TRUE (Low level) and sends the numerics given to the GPIB. The numeric is converted into an 8-bit binary data and output to the GPIB. Therefore, the numerics to be used are the range of 0 to 255 and the numerics of decimal point e expression are automatically converted into integers.

DATA: Sets the ANT line to FALSE (High level) and sends the numerics given to the GPIB. The numerics to be used are the same as CMD.

EOI: Outputs the single line signal EOI. It should be specified after the DATA statement.

LISTEN: Sends the numerics given to the GPIB as listener address group (LAG). Multiple numerics can be specified.

TALK: Sends the numerics given to the GPIB as talker address group (TAG). Multiple numerics cannot be specified.

UNT: Sends the UNT command to the GPIB. The talker (unit specified as talker before executing this command) can be canceled.

UNT: Sends the UNL command to the GPIB. The listener (unit specified as listener before executing this command) can be canceled.

Example

```
10 SEND UNT UNL LISTEN 1, 2, 3 TALK 4
20 SEND UNT CMD 63, 33 DATA 30,54
```

Note

The SEND statement is not available in the ADDRESSABLE mode.

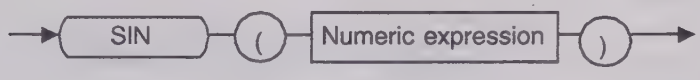
66. SIN

Outline

The SIN is a built-in function that is used to return the sine value. The unit of the value specified with the numeric expression should be in radian.

Syntax

(1)-1



(1)-2

SIN (numeric expression) →

Example

```
PRINT SIN(PI/2)
A=SIN(B)
```

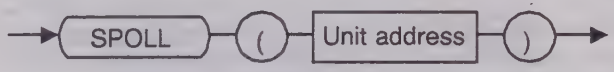
67. SPOLL

Outline

The SPOLL statement is used to perform the serial polling of the specified unit and to read out the status byte.

Syntax

(1)-1



(1)-2

SPOLL (unit address)

Description

- When the analyzer is set to the SYSTEM CONTROLLER mode, the SPOLL statement executes the serial polling for the other GPIB units.
- When the unit address is 0 to 30, the SPOLL statement executes the serial polling for the units corresponding to each address.
- When the unit address is 31, the SPOOL statement retrieves the status byte for the analyzer regardless of whether ?>the analyzer<? is set to the SYSTEM CONTROLLER mode or the ADDRESSABLE mode.

Example

Searches the internal status after the sweep end.

```
10  INTEGER Sp
20  OUTPUT 31;''IP''
30  OUTPUT 31;''CF30MZ SP100MZ SW2SC''
40  !
50  OUTPUT 31;''*CLS''      ! Status byte clear
60  OUTPUT 31;''OPR8''      ! Enables the sweep end bit
70  OUTPUT 31;''*SRE128''   ! Operation status enable
80  OUTPUT 31;''SO''        ! SRQ enable
90  ON ISRQ GOTO 130
100 ENABLE INTR
110 GOTO 110
120 !
130 Sp=SPOLL(31)
140 PRINT ''STATUS = '' ;Sp
150 END

<Execution result>
 STATUS = 192
```

Note

In the ADDRESSABLE mode, if the unit address between 0 to 30 is specified and the SPOLL is executed, the value "0" will be returned.

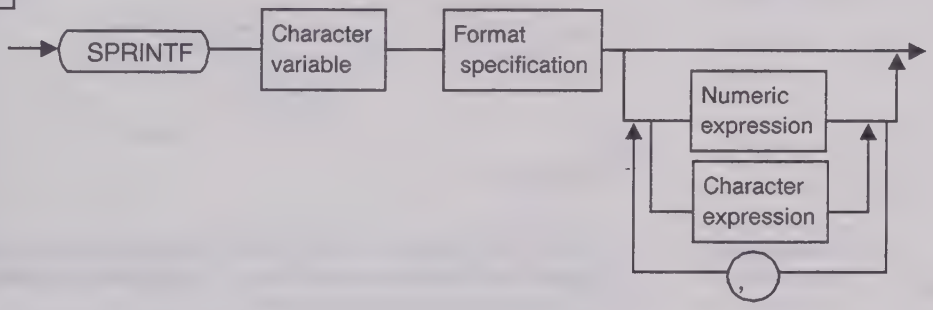
68. SPRINTF

Outline

The SPRINTF statement is used to convert the format in accordance with the format conversion of the PRINTF command and to assign the result to the character-string variable.

Syntax

(1)-1



(1)-2

SPRINTF character-string variable format specification [numeric expression | character expression {, numeric expression | character expression}]

Description

- The SPRINTF statement converts the expression value in accordance with the format conversion of the PRINTF command, and assigns the result to the character-string variable of first parameter.
- Pay attention to the format specification, the number of expression, and the character-string variable size for storing the result.
  If the character string for storing the result does not have enough capacity (free space), the BASIC buffer may be damaged.

  The method of format specification is refer to "57. PRINTF" of section 4.3.

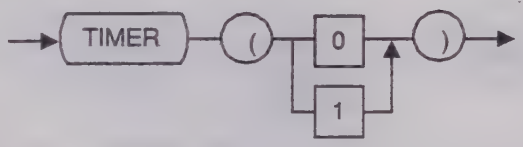
69. TIMER

Outline

The TIMER statement is used to read/reset the internal system time.

Syntax

(1)-1



(1)-2

TIMER (0 | 1)

Description

- The TIMER statement is the built-in function, which returns the internal system time with the unit of sec. This function is mainly used to check the measurement operation time.
  When the argument 0 is specified: Reads out the internal system time.
  When the argument 1 is specified: Resets the internal system time.
- The read out value with the resolution of 10msec includes an error of ±10msec.

Example

```
10  INTEGER I
20  TIMER(1)
30  FOR I=0 TO 10000
40  NEXT I
50  T1=TIMER(0)
60  !
70  TIMER(1)
80  FOR I=0 TO 10000
90    PRINT I
100 NEXT I
110 T2=TIMER(0)
120 !
130 PRINT "PRINT Command execute time is " ;T2-T1
140 STOP
```

70. TIME$

Outline

The TIME$ statement is used to read/set the time of the built-in timer.

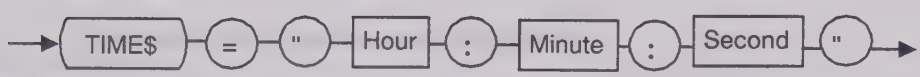
Syntax

(1)-1

(1)-2

TIME$

(2)



(2)-2

TIME$ = "hour : minute : second"

Description

- The TIME$ statement reads out the time of the built-in timer (RTC).
- The TIME$ statement can change the time which is read out.
  Input as follows:

```
TIME$="23:43:12"
TIME$="11:5:6"
```

Example

```
10 DIM T$[10]
20 T$=TIME$
30 PRINT "Time is "; T$
40 PRINT "Time Reset"
50 TIME$="0:0:0"
60 STOP
```

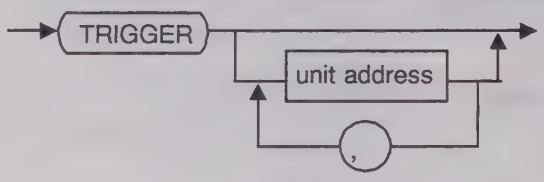
71. TRIGGER

Outline

The TRIGGER statement is used to send the group execute trigger (GET) of address command group (ACG) to the all units connected to the GPIB or to the particular unit selected.

Syntax

(1)-1



(1)-2

TRIGGER [unit address {, unit address }]

Description

- If only the TRIGGER statement is executed without specifying the unit address, only the the group execute trigger (GET) of address command will be transmitted. In this case, the unit to be triggered must be set as listener in advance.
- If the unit address followed after the TRIGGER statement is specified, the GET command will be transmitted to only the unit address specified by its unit address.

Example

```
10 TRIGGER 1
20 TRIGGER
```

Note

The TRIGGER statement is not available in the ADDRESSABLE mode.

72. WAIT

Outline

The WAIT statement is used to wait for the specified time.

Syntax

(1)-1

(1)-2

WAIT time

Description

- The WAIT statement waits for the specified time. The unit of time is msec. The setting range of time between 0 to 65535.

Example

```
10 INTEGER T
20 T=30
30 PRINT T;"[msec] Wait !!"
40 WAIT T
50 STOP
```

73. WAIT EVENT

Outline

The WAIT EVENT statement is used to wait the event until the specified event is generated.

Syntax

(1)-1

(1)-2

WAIT EVENT event number

Description

WAIT EVENT 1 → Waits until the sweep end event is generated.

Example

Searches the internal status after the sweep end.

```
10 OUTPUT 31;''IP''
20 OUTPUT 31;''CF30MZ SP100MZ SW2SC''
30 !
40 OUTPUT 31;''*CLS''      !  Status byte clear
50 OUTPUT 31;''OPR8''      !  Enables the sweep end bit
60 OUTPUT 31;''*SRE128''   !  Operation status enable
70 WAIT EVENT 1
80 PRINT ''sweep end''
90 END

<Execution result>
 sweep end
```

5. ERROR MESSAGES

5.1 How to Check Error Message Line Number

When the PRINT WRRM$(0) is executed, the last error message (including the line number in case of the error during the program execution) will be displayed.

5.2 How to Check Program Current Position

The symbol"@" is a system variable that is used to check the current line number of the program being executed.

Example: PRINT @

5.3 Error Message List

- The error messages are described in the following table in the order of error class (error number). And in these error messages, character strings are explained as xxx and numerics are as yyy.

- Error class : 1; Data input
  2; Data calculation processing
  3; BASIC syntax
  5; Others

(1 of 6)

Error class (Error number)	Error message	Description
1(1)	xxx1(xxx2) error	xxx1 command is not available for xxx2 file.
1(2)	xxx1(xxx2, xxx3) error	xxx1 command is not available for xxx2 file and xxx3 file.
1(3)	Memory space full	The heap memory space for the BASIC is insufficient.
1(4)	Bad free call	A defective condition is detected when the memory is freed up.
1(5)	File format error	There are no terminator within 256 characters.
1(6)	File is NOT open	The file has not been registered in the specified descriptor. (The file has not been opened.)
1(7)	Cannot read from "xxx" file	The specified character number could not be read from xxx file.

(2 of 6)

Error class (Error number)	Error message	Description
1(8)	Cannot write to "xxx" file.	Data can not be written to xxx file.
1(9)	"xxx" file cannot be opened	"xxx" file cannot be opened because it does not exist.
1(10)	Only one OUTPUT file can be opened	Two or more files were tried to open with the OUTPUT mode.
1(11)	Only one INPUT file can be opened	Two or more files were tried to open with the INPUT mode.
1(12)	xxx: "xxx" file was opened with xxx mode	"xxx" file was accessed with the different mode from the mode with which it had been opened.
1(13)	"xxx" file is already opened	The file already opened was tried to open again.
1(14)	End of "xxx" file	Data was tried to read after EOF(End Of File) was detected.
1(15)	Abort	The GPIB control statement was aborted in the execution, or an error occurred on the GPIB bus.
1(16)	Time out	The GPIB time-out occurred.
1(17)	GPIB syntax error	The GPIB command is incorrect.
1(18)	Missing GPIB code	Cannot be assigned into the character variable.
1(19)	yyy: Unit address error in xxx	The GPIB address is incorrectly specified in xxx command.
1(20)	Device not ready	Device is not ready in the specified drive.
1(21)	Invalid character	The character that has been specified is incorrect.
1(22)	Device not found	The specified device is not found.
1(23)	Too many dots	There are too many "." (CHDIR command).
1(24)	Root has no parent	The parent directory that has specified is not found.

(3 of 6)

Error class (Error number)	Error message	Description
1(25)	No such file or directory	The specified file or directory is not found.
1(26)	Already 8 files are opened	The eight files have already been opened.
1(27)	Cannot execute to protected file	The access to the protected file cannot be made.
1(28)	Illegal file name	The protected file was tried to be copied or renamed.
2(30)	Zero divide	0 division was executed.
2(31)	Substring error	Substring is incorrectly specified.
2(32)	Overflow value	The value of operation exceeded the allowable range.
2(33)	Invalid string constant	There is no character in " ".
2(34)	String length is too long	Declaration of character string variable exceeded the maximum value (128character).
2(35)	xxx: Cannot convert into string	The character cannot be converted into ASCII has detected in the string.
3(40)	Program is NOT exist	Executed the program not exist.
3(41)	xxx: Syntax error	The incorrect syntax is used. (Fatal error)
3(42)	Expression format error	Expression formatted incorrectly. (Specification of the arguments, separators and so on.)
3(43)	Parameter error	Parameter is not set correctly about its format or type.
3(44)	xxx2: Invalid type in xxx1	xxx1 contains the invalid type (xxx2).
3(45)	xxx2: Invalid first type in xxx1	The first part of command syntax (xxx2) is incorrect.
3(46)	xxx2: Invalid second type in xxx1	The second part of command syntax (xxx2) is incorrect.
3(47)	Cannot assigned into this token	Cannot be assigned into the specified character string variable.

(4 of 6)

Error class (Error number)	Error message	Description
3(48)	NO operand in xxx	Operation format for xxx was set incorrectly.
3(49)	Not found DATA statement	DATA statement is not found in the direction of RESTORE.
3(50)	Unmatched DATA's values and READ variable	The value specified by the DATA statement is unmatched with the data read in READ statement. (Data read in READ statement does not exist).
3(51)	Unmatched IMAGE-spec in USING	Specification of IMAGE in USING is unmatched.
3(52)	Not found THEN in IF	THEN is not found after IF statement.
3(53)	FOR's nest is abnormal.	Nesting to FOR statement could not execute properly.
3(54)	FOR variable does NOT exist.	The counter variable of FOR statement does not exit.
3(54)	NEXT variable does NOT exist.	The counter variable of NEXT statement does not exit.
3(55)	FOR <init value> does NOT exist.	The initial value of FOR statement does not exist.
3(56)	Unbalanced FOR variable in NEXT	Relation between For statement and NEXT statement is not normal.
3(57)	Unbalanced NEXT statement	NEXT statement does not exist even the existence of FOR statement.
3(58)	Unbalanced BREAK	BREAK statement does not exist between FOR statement and NEXT statement.
3(59)	SELECT nesting overflow	Nesting to SELECT statement exceeded the capacity.
3(60)	GOSUB nest overflow	Nesting between GOSUB statement and RETURN statement exceeded the capacity.
3(61)	Label "xxx" is already exists.	Label that has already existed is tried to be registered.

(5 of 6)

Error class (Error number)	Error message	Description
3(62)	Unbalanced xxx	Statement construction of xxx is not balanced.
3(63)	Unbalanced xxx block	xxx block (IF, SELECT and so on) is incorrect.
3(64)	Invalid dimension parameter	Parameter of an array variable is not correct.
3(65)	Array's range error	Subscript of the array variable is out of declaration range.
3(66)	Array's declaration error	Format or method of use of array variable is incorrect.
3(67)	Unbalanced line No.	Specification of the line number is incorrect.
3(68)	xxx: Reserved value	A value was tried to be assigned after the system variables and so on had been reserved.
3(69)	Undefined label	Specified label does not existed.
3(70)	Label not found	Specified label does not existed.
3(71)	Unknown line No.	Specified line does not existed.
3(72)	Not found line No.yyy	Line No. yyy does not existed.
3(73)	Line No.yyy is out of range	The line number was specified in minus or exceeding 65535.
3(74)	Cannot move line	The line number cannot be changed by the REN command.
3(75)	xxx error(s) appeared	The total number of errors in execution of the program is appeared.
3(76)	Not available ASCII char(yyy)	Invalid ASCII character yyy (in hexadecimal) is detected.
3(77)	Program mistaked at xxx	An error is in the program.
3(78)	xxx: Undefined icode	Invalid intermediate code was detected.
3(79)	Undeclare value	Array variable was tried to be used without the declaration.
3(80)	Cannot specify USING	USING cannot be specified by the specified file type.

(6 of 6)

Error class (Error number)	Error message	Description
3(81)	Program cannot changed	Program was tried to be changed in the execution of the program.
3(82)	Uninstalled type (xxx)	Variable is incorrectly formatted.
3(83)	Parameter is out of range	The argument out of range was passed.
3(84)	Merge program is NOT exist	Though MERGE did not exist, MEARE DEL was executed.
3(90)	Program cannot be continued.	The terminated program was tried to restart again.
3(91)	Warning: cannot execute 'xxx'	In the condition that "xxx" command cannot be executed.
3(92)	Warning: cannot execute 'xxx' (yyy)	"xxx" command was tried to be executed in the execution of "yyy".
3(93)	Cannot execute 'xxx'(yyy)	Cannot execute 'xxx' (yyy): "xxx" command was tried to be executed in "yyy" condition.
	Unmatched FOR to NEXT variable	Unmatched FOR to NEXT variable: The counter variables between FOR statement and NEXT statement are unmatched.
	Unmatched number of FOR to NEXT	Unmatched number of FOR to NEXT: Nesting of FOR to NEXT statement is incorrect.

ALPHABETICAL INDEX

[A]

ABS 4-21
ADDRESSABLE mode 1-2
Alphanumeric Characters 4-5
ATN 4-22

[B]

BASIC COMMANDS 3-1
BASIC STATEMENT 4-1

[C]

CAT 3-5
Changing File Name 2-9
CHDIR 3-6
CHKDSK 3-7
CHR$ 4-23
CLEAR 4-24
CLOSE 4-25
CLS 4-26
COLOR 4-27
Command and statement syntax 1-1
Command Grammar and Application .. 3-5
COMMON 4-28
Connecting External Keyboard 2-13
CONSOLE 4-30
CONT 3-8
CONTROL 3-9
COPY 3-11
Correspondence Table between Full Name and Short Name 4-2
COS 4-31
CSRLIN 4-32
CSRPOS 4-33
CURSOR 4-34

[D]

DATA 4-35
Data Input Keys 2-2
DATE$ 4-36
DEL 3-12
DELIMITER 4-37
DIM 4-38
DISABLE INTR 4-40
Drive Slot for Memory Card 2-5
DSTAT 4-41

[E]

ENABLE INTR 4-43
END 3-13
ENTER 4-44
ENTER USING 4-47
Erasing Files 2-9
ERRM$ 4-50
ERRN 4-51
Error Message List 5-1
ERROR MESSAGES 5-1
EXP 4-52
External Keyboard 2-12

[F]

File Management 2-7
File Management 2-6
FOR - TO - STEP, NEXT, BREAK, CONTINUE 4-53
FRE 4-55
Function Keys 2-2

[G]

GLIST 3-14
GLISTN 3-15
GOSUB, RETURN 4-56
GOTO 4-58
GPIB mode 1-2
GPRINT 4-59

[H]

How to Check Error Message Line Number 5-1
How to Check Program Current Position 5-1

[I]

IF-THEN, ELSE, END IF 4-61
INDENT 3-16
INKEY$ 4-64
INPUT (INP) 4-65
Inputting, Executing and Ending Program 2-1
Insertion and Ejection of Memory Card 2-5
INTEGER 4-67

INTERFACE CLEAR 4-69
INTRODUCTION 1-1

[K]

Key Word List 4-2

[L]

LEN 4-70
LIST 3-17
List of Command Function 3-2
List of Command Syntax 3-3
LISTN 3-18
LLIST 3-19
LLISTN 3-20
LOAD 3-21
Loading Files 2-8
LOCAL 4-71
LOCAL LOCKOUT 4-72
LOG 4-73
LPRINT 4-74

[M]

Memory Card 2-3
Memory Card Specifications 2-4
MERGE 3-22

[N]

Note on Handling the Memory Card .. 2-4
NUM 4-75

[O]

Object 4-3
OFF END 4-76
OFF ERROR 4-77
OFF KEY 4-78
OFF SRQ, OFF ISRQ 4-79
ON DELAY 4-80
ON END 4-81
ON ERROR 4-82
ON KEY 4-83
ON SRQ, ON ISRQ 4-85
OPEN 4-86
OPERATING BASICS 2-1
Operators 4-8
OUTPUT 4-88
OUTPUT USING 4-91
Overview of Operation 2-1

[P]

Panel Operation 2-1
PAUSE 3-24
POS 4-93
Precautions Common to All Commands 3-4
PRINT [USING] 4-94
PRINTER 4-97
PRINTER 3-25
PRINTF 4-98
Program Input 3-5
Program Structure 4-1
Programming Rules 4-1
PURGE 3-25
PWD 3-26

[R]

READ 4-100
REAL 4-101
REM 4-102
REMOTE 4-103
REN 3-27
RENAME 3-28
REQUEST 4-104
RESTORE 4-105
RUN 3-29

[S]

SAVE 3-30
SCRATCH 3-31
Screen Layout 2-10
SELECT, CASE, ENS SELECT 4-106
SEND 4-107
SIN 4-109
SPOLL 4-110
SPRINTF 4-111
Statement Function List 4-12
Statement Syntax and Use 4-21
Statement Syntax List 4-14
STEP 3-32
STOP 3-33

Storing Files 2-8
Summary 2-6
SYSTEM CONTROLLER mode 1-2

[T]

TIME$ 4-113
TIMER 4-112
TRIGGER 4-114

[U]

Usable Memory Card 2-3

[V]

Various Commands 3-1
Various Statements 4-12

[W]

WAIT 4-115
WAIT EVENT 4-116

WARRANTY

ADVANTEST product is warranted against defects in material and workmanship for a period of one year from the date of delivery to original buyer.

LIMITATION OF WARRANTY

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by buyer, unauthorized modification or misuse, accident or abnormal conditions of operations.

No other warranty is expressed or implied. ADVANTEST specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

ADVANTEST shall not be liable for any special incidental or consequential damages, whether in contract, tort or otherwise.

Any and all warranties are revoked if the product is removed from the country in which it was originally purchased.

SERVICE

During the warranty period, ADVANTEST will, at its option, either repair or replace products which prove to be defective.

When trouble occurs, buyer should contact his local supplier or ADVANTEST giving full details of the problem and the model name and serial number.

For the products returned to ADVANTEST for warranty service, buyer shall prepay shipping and transportation charges to ADVANTEST and ADVANTEST shall pay shipping and transportation charges to return the product to buyer. However, buyer shall pay all charges, duties, and taxes incurred in his country for products returned from ADVANTEST.

CLAIM FOR DAMAGE IN SHIPMENT TO ORIGINAL BUYER

The product should be throughly inspected immediately upon original derivery to buyer. All material in the container should be checked against the enclosed packing list or the instruction manual alternatively. ADVANTEST will not be responsible for shortage unless notified immediately.

If the product is damaged in any way, a claim should be filed by the buyer with carrier immediately. (To obtain a quotation to repair shipment damage, contact ADVANTEST or the local supplier.) Final claim and negotiations with the carrier must be completed by buyer.

SALES & SUPPORT OFFICES

Advantest Singapore Pte. Ltd.
2 Corporation Road #06-03/04, Corporation Place, Singapore 618494
Phone : 65-261-8366 Facsimile : 65-261-8377

ROHDE & SCHWARZ
Engineering and Sales GmbH, Mühldorfstr. 15 (P.O.B. 80 1429, D-81614 Munich)
D-81671 München
Phone : 49-89-4129-3711 Facsimile : 49-89-4129-3723

TEKTRONIX INC.
P.O. Box 500, M/S 39-520, Beaverton, Oregon 97077-0001
or
Howard Vollum Industrial Park, M/S 58-743, Beaverton, OR, 97077, U.S.A.
Inside the U.S. 1-800-426-2200
Outside the U.S. 1-503-627-1933

Technology Support on the Leading Edge
ADVANTEST®
ADVANTEST CORPORATION
Shinjuku-NS Building, 4-1, Nishi-Shinjuku 2-chome, Shinjuku-ku, Tokyo 163-08, JAPAN
Phone:(03)3342-7500 Facsimile:(03)3342-7410 Telex:232-4914 ADVAN J

ADVANTEST

ADVANTEST CORPORATION

www.manualsplus.com

1006430